# NetFusion Software

## *uCLinux User Guide*

# NetFusion Software uCLinux User Guide

V1.0 - August 2014

## Table of Contents

**1 INTRODUCTION TO NETFUSION ..... 7**

1.1 DEVELOPER SIMPLICITY ..... 7
1.2 UCLINUX ..... 8
1.3 NETFUSION UCLINUX EXTENSION ..... 8
1.4 UCLINUX CORTEX-M ..... 8

**2 HARDWARE PLATFORM ..... 9**

**3 PRE-INSTALLED DEMO ..... 10**

3.1 PRE-INSTALLED DEMO ON THE NETFUSION HARDWARE PLATFORM ..... 10
3.2 DESCRIPTION OF THE PRE-INSTALLED DEMO ..... 10

**4 SOFTWARE FUNCTIONALITY ..... 15**

4.1 SUPPORTED FEATURES ..... 15
4.1.1 U-boot firmware ..... 15
4.1.2 uCLinux ..... 15
4.1.3 NetFusion UIO Drivers, Additional Packages ..... 16
4.1.4 Development tools ..... 16
4.1.5 Development environment ..... 16
4.2 KNOWN PROBLEMS & LIMITATIONS ..... 16

**5 CONFIGURING STARTER KIT FOR UCLINUX DEVELOPMENT ..... 17**

5.1 POWERING UP AND CONNECTING THE SERIAL CONSOLE DEVICE ..... 17
5.2 CONNECTING A SERIAL CONSOLE TERMINAL APPLICATION TO THE M2S-FG484 SOM NETFUSION PCB 18
5.3 INSTALLING U-BOOT TO THE M2S-FG484 SOM OF NETFUSION ..... 19
5.4 INSTALLING THE TFTP SERVER ..... 20
5.5 BOOTING THE UCLINUX IMAGE ..... 20

**6 SMARTFUSION2 SOM BOARD UCLINUX SOFTWARE SET-UP ..... 22**

6.1 UCLINUX DEMO AND DEVELOPMENT ENVIRONMENT ..... 22
6.2 U-BOOT ENVIRONMENT ..... 22
6.3 ETHERNET MAC ADDRESS ..... 23
6.4 NETWORK CONFIGURATION ..... 23
6.5 RUNNING PRE-INSTALLED UCLINUX IMAGE ..... 23
6.6 LOADING UCLINUX IMAGES ..... 23
6.7 U-BOOT BUILD ..... 25
6.8 U-BOOT INSTALLATION ..... 25

**7 UCLINUX CORTEX-M SOFTWARE DEVELOPMENT ENVIRONMENT ..... 26**

7.1 DISTRIBUTION AND INSTALLATION ..... 26
7.1.1 Distribution Image ..... 26
7.1.2 Installation Tree ..... 26
7.1.3 GNU Cross-Build Tools ..... 27
7.1.4 Activation ..... 28
7.1.5 Dependency on NetFusion Components ..... 28
7.2 PROJECTS FRAMEWORK ..... 29
7.2.1 Multiple Target Projects ..... 29

7.2.2 Hello World Demo Project ..... 30

**8 U-BOOT LOADER FACILITY ..... 31**

8.1 POINTERS TO DETAILED INFORMATION ON U-BOOT ..... 31
8.2 U-BOOT: THEORY OF OPERATION ..... 31
8.3 SELECTED USE SCENARIOS ..... 32
8.3.1 U-boot Environment ..... 32
8.3.2 Autoboot ..... 33
8.3.3 Networking in U-boot ..... 33
8.3.4 Flash-related Commands ..... 33
8.3.5 U-boot Self-Upgrade ..... 35

**9 SAMPLE DEVELOPMENT SESSION ..... 37**

9.1 SAMPLE SESSION OUTLINE ..... 37
9.2 CREATE A NEW PROJECT ..... 37
9.3 SET UP TARGET FOR DEBUGGING OF THE NEW PROJECT ..... 37
9.4 UPDATE THE NEW PROJECT ..... 39
9.5 INSTALL THE NEW PROJECT TO FLASH ..... 40
9.6 DEVELOP A CUSTOM LOADABLE DEVICE DRIVER OVER NFS ..... 41
9.7 DEVELOP A CUSTOM USER-SPACE APPLICATION OVER NFS ..... 43

**10 BUILDING UCLIBC ..... 44**

**11 BUILDING MULTI-THREADED APPLICATIONS ..... 47**

**12 NETFUSION PROJECT IN THE UCLINUX TREE ..... 48**

12.1 WHAT IS THE NETFUSION PROJECT WITH-IN THE UCLINUX? ..... 48
12.2 SOURCE FILE STRUCTURE & NETFUSION PACKAGE ..... 49
12.2.1 Package Deliverable ..... 49
12.2.2 Host PC Cross-Compile Directory Structure ..... 50
12.2.1 File List of *mtiptest* Directory ..... 56
12.2.2 SQLite Directory ..... 59
12.3 SQLITE USAGE ..... 60
12.4 THE AMBA DMA CONTROLLER DEVICE DRIVER ..... 60
12.4.1 What is the AMBA DMA Controller? ..... 60
12.4.2 Source Files ..... 60
12.4.3 Concept ..... 61
12.4.4 Driver Operation & Functional Description ..... 62
12.4.5 High and Low Priority Channels ..... 62
12.4.6 Linux/uC-Linux Network Interface Exposure ..... 62
12.4.7 DMA Controller Device Driver Parameter List ..... 63
12.5 BUILDING THE NETFUSION CUSTOMIZED UCLINUX KERNEL ..... 63
12.5.1 Building the SQLite Incrementally ..... 64
12.5.2 Building the main uCLinux with NetFusion Incrementally ..... 64
12.5.3 Uploading the kernel to the NetFusion ..... 64
12.6 BOARD UCLINUX TARGET DIRECTORY STRUCTURE ..... 65
12.6.1 NetFusion Web Server ..... 69

**13 AFTER FIRST COMPILATION, GETTING THE SYSTEM CUSTOMIZED FOR THE USER 71**

**14 NETFUSION BOARD DRIVERS ..... 73**

14.1 UCLINUX APPLICATION UIO ..... 73
14.2 NETFUSION USER-SPACE DRIVER LIST ..... 73

14.3 Option Selector Switches ..... 74
14.4 RS232 / GPIO Extended Signals ..... 74
14.5 Relays and the Selector Switches ..... 74
14.6 Relay Output Connector ..... 75
14.7 LCD Connector ..... 75
14.8 Analogue Contact Input Connector ..... 76
14.9 RS232 UART Serial Connectors ..... 76
14.10 RS485 UART Connector ..... 77
14.11 GPIO Connector ..... 77
14.12 Stereo Audio Phono-Jacks ..... 78
14.13 NetFusion Hardware Drivers Usage ..... 79
14.13.1 Analogue Contacts (ACI) ..... 79
14.13.2 RS485 UART ..... 80
14.13.3 Real-Time Clock (RTC) ..... 81
14.13.4 LCD Controller ..... 82
14.13.5 General I/O (GPIO) ..... 83
14.13.6 Line-OUT Stereo Audio ..... 84
14.13.7 Line-IN Stereo Audio ..... 85
14.13.8 PCB Temperature Sensor ..... 86
14.13.9 Voltage PCB Supervisor ..... 87

**15 RUNNING A SOFTCONSOLE "HELLO, WORLD!" PROJECT ON NETFUSION ..... 88**

15.1 Hardware Platform ..... 88
15.2 Installing the Software on the Development Host PC ..... 88
15.3 Running and Debugging the Project ..... 89
15.4 Modifying the Libero and SoftConsole Projects ..... 89

**16 INSTALLING A BOOTABLE UCLINUX IMAGE TO THE NETFUSION BOARD ..... 91**

**17 BOOTING UCLINUX ON SMARTFUSION2 IN 0.5 SECONDS ..... 92**

17.1 Fast-booting uCLinux on SmartFusion2 ..... 92
17.2 uCLinux Bootstrap Sequence ..... 92
17.3 Installing the Demo ..... 93
17.4 Understanding the Demo ..... 93
17.5 Configuring for Fast-Boot ..... 96
17.6 Restoring Default Configuration ..... 98

**18 USING DHCP CLIENT IN U-BOOT ..... 99**

**19 SETTING THE STACK SIZE FOR A USER-SPACE APPLICATION ..... 100**

**20 BUILDING MULTI-THREADED APPLICATIONS ..... 102**

**21 HOW TO DEFINE AN IRQ HANDLER IN A CUSTOM UCLINUX DEVICE DRIVER ..... 103**

**22 UNDERSTANDING WHAT CAUSES SEGV IN AN APPLICATION ..... 105**

**23 REFERENCES ..... 106**

**24 CONTACT ..... 107**

**25 DOCUMENT HISTORY ..... 108**

# List of Figures

Figure 1 - The NetFusion-CAT5 PCB Hardware Platform (SERDES variant also available) ..... 9
Figure 2 - USB Cable Connected to the USB-UART Interface from a PC ..... 17
Figure 3 - TFTP Application on Windows XP/7/8 ..... 20
Figure 4 – An example on Ubuntu Linux of the Terminal being launched ..... 48
Figure 5 – Default MAC Setting ..... 58
Figure 6 – Default Server Name Setting ..... 58
Figure 7 - Main opening screen for the NetFusion Demo Website ..... 69
Figure 8 – Drivers/Hardware Page for the NetFusion Demo Website ..... 70

# List of Tables

Table 1: App Program File List and Meanings ..... 56
Table 2: App Program Exceptions for File List ..... 57
Table 3: App Program Source content ..... 57
Table 4: App Program Source important variables ..... 58
Table 5 : Full NetFusion User Space Hardware Drivers ..... 73
Table 6: Document History Entry Log ..... 108

# About This Document

This specification fully documents and describes the process of file structure of source, compiling, building, kernel building, scripts, toolset description, API, coding and using the board drivers for the NetFusion baseboard. The deliverable bundle is the NetFusion software project as an extension of the Emcraft uCLinux installed directory tree.

# Intended Audience

This document is fully written for Nine Ways' customers using the hardware NetFusion-CAT5/SFP baseboard for evaluation, full product deployment, test and/or further customer product development. This document covers all variants of the NetFusion family as the Ethernet media hardware derivative are abstracted at the uCLinux/Software level.

# 1 Introduction to NetFusion

The NetFusion PCB baseboard forms several different product usages. Simplistically, it is an interface product for Emcraft's SOM-F484 series SmartFusion 2 FPGA sub-system to allow real-world connectivity in a deployed light/heavy industrial environment in any situation. It incorporates not only many different legacy connectors and interfaces but also compact high speed connectivity such as 10/100/1000M Ethernet, USB and GPIO. Its versatility enables it to be an ideal platform as a conduit for the highly reputable and endurable M2S-SOM-484 family of SmartFusion2 FPGA system-on-module to interact in many given environments and deployable scenarios. On their own, the SOM FPGA units have no housing or any means to survive alone and must be hosted on a customized baseboard to allow the devices to perform the tasks that developers' desire. NetFusion short cuts the development process in achieving a customized baseboard as much as possible with exceptions only when extremely bespoke design requirements are needed.

The product can be used either in the format of an IP core Ethernet SWITCH development kit (as most of the hardware orientates network communications - hence the name NetFusion), or as an M2S-SOM SmartFusion2 trial and evaluation baseboard (more extensive interfaces than the more basic competitors), or finally as a fully ready-to-deploy industrial product solution to a customer's needs having developed customized software and IP core designs to tailor to intended product functionality.

NetFusion is ideal for collating IP cameras, old legacy equipment using RS232, 485 or contact relays into an Ethernet topology at an industrial site, building or remote location. Moreover, with a collection of modern communication interfaces, the seemingly huge number of variations of product usage allows this to be a truly powerful, attractive, efficient and desirable embedded tool to add to your portfolio.

## 1.1 Developer Simplicity

With the PCB hardware already available for you to evaluate, develop and then deploy to your customers, it can be as easy as developing application programs within the uCLinux software environment and if needed, adding IP cores into the Libero IDE fabric design. In short, design development time can be as little as months to tailor and customize functionality to your needs. If you add an IP core, you can write the device driver for uCLinux then test and debug. With a full UNIX file system, using the bundled SQLite (MySQL-uCLinux) provides easy database management at the power your embedded device.

## 1.2 uCLinux

uCLinux on SmartFusion2 opens up a new horizon of possibilities for application designers. SmartFusion2, with its ARM Cortex-M3 processor core and I/O interfaces, FPGA and programmable analogue, is truly feature-rich. All these various features must be supported with software drivers, stacks and programming interfaces. uCLinux, undoubtedly, is the only OS that is ready for the job today. Many hardware interfaces of SmartFusion2 are supported out-of-box by standard uCLinux device drivers and software frameworks. For those that are not, uCLinux provides a convenient development environment as well as an array of open-source projects that can be used as starting points when developing support for required hardware interfaces.

uCLinux experts are everywhere, and support is easy to find. Whatever new feature or capability may be required by your customers in the future, it is a safe bet that uCLinux will provide a level of support for it. External memory utilized by uCLinux takes away the worries of having to squeeze required software functionality into the limited space of the integrated eNVM and eSRAM of SmartFusion2. Products based on uCLinux can be made feature-rich and highly user-friendly. Reliable upgrades for new releases and capabilities are easy to implement, providing for obvious maintenance procedures and a long lifespan for your designs as well as your products.

## 1.3 NetFusion uCLinux Extension

The Emcraft M2S-SOM-F484 plug-in board utilizes uCLinux as standard. However, in order for NetFusion to build on this foundation, Nine Ways has developed UIO user space device drivers to access and control all PCB hardware that is not specifically any of the Ethernet pathways in the fabric of the SmartFusion2 FPGA. All are fully available as open source in your GZIP NetFusion software bundle and will be deployed as binary in the UNIX embedded file-system.

## 1.4 uCLinux Cortex-M

The aforementioned uCLinux mentioned is based on the uCLinux Cortex-M and provides a platform and software development environment for evaluation and development of uCLinux on the Cortex-M CPU core on the MCU devices. This is the chosen OS platform for the NetFusion software environment running on the ARM Cortex-M3 processor inside the SmartFusion2 FPGA.

It is fully available as open-source and can be changed and re-built to suit your needs. No royalty rights are involved and it is free to use and deploy.

## 2 Hardware Platform

For this specific product, uCLinux Cortex-M comes as a software distribution image supporting the corresponding ARM Cortex-M microcontroller embedded in the SmartFusion2 FPGA. It provides support for the hardware architecture on the SOM and comes as a Board Support Package (BSP) specific to the NetFusion board.

This User's Manual document proceeds to outline information that is common for all supported MCU architectures but in particular to this NetFusion board.

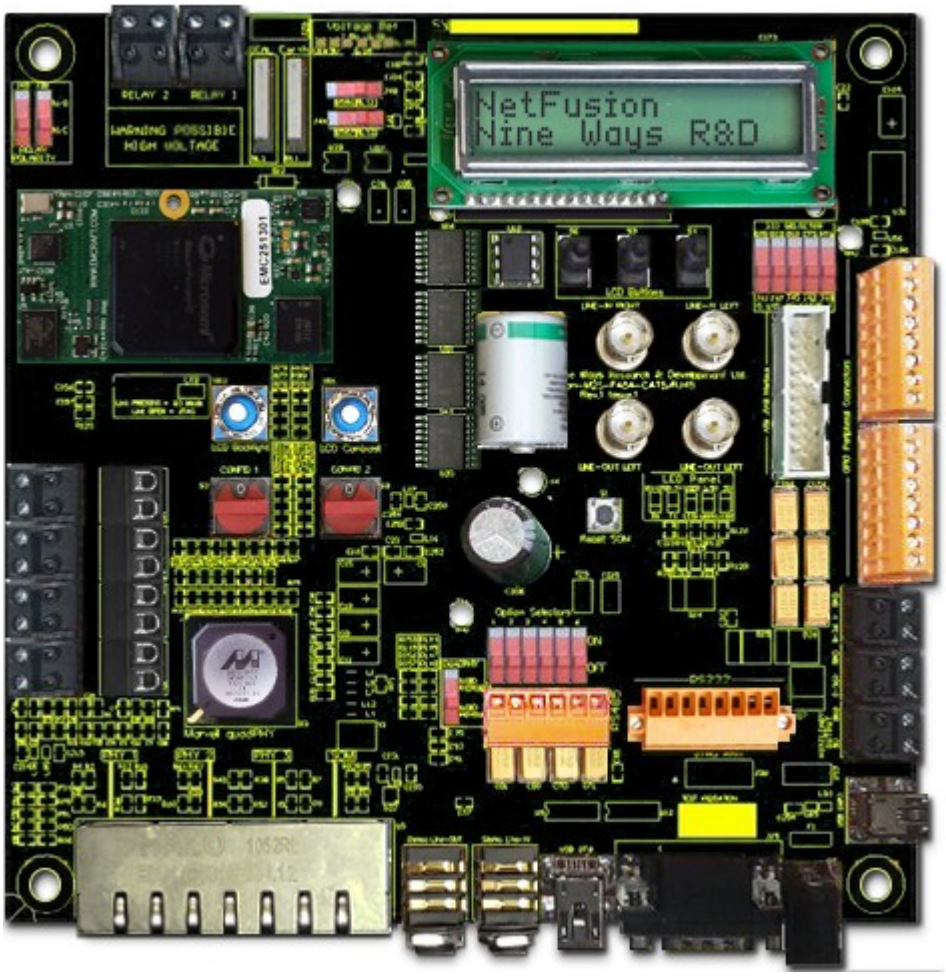



**Figure 1 - The NetFusion-CAT5 PCB Hardware Platform (SERDES variant also available)**

# 3 Pre-installed Demo

## 3.1 Pre-installed Demo on the NetFusion Hardware Platform

NetFusion comes with the U-boot firmware installed into the internal flash of the microcontroller device and uCLinux image installed into the external flash memory.

## 3.2 Description of the Pre-installed Demo

The installed uCLinux image provides a default NetFusion start-up environment that includes all SOM hardware and the NetFusion baseboard hardware support drivers.

On a power-on or reset, U-boot runs from the embedded flash and uses the embedded SRAM of the microcontroller as storage for volatile data.

Having completed the basic initialization, U-boot configures the memory controller to allow accesses to the external RAM and flash memory and then copies the uCLinux image from flash to RAM and jumps to the uCLinux kernel entry point in RAM. It is possible to interrupt the U-boot autoboot sequence by hitting a key before U-boot starts relocating the uCLinux image to RAM and enter the command line interface of U-boot, however assuming no operator intervention; U-boot proceeds to boot uCLinux up as soon as possible:

```
U-boot 2010.03-linux-cortexm-1.12.0 (Dec 06 2013 - 19:43:45)

CPU: SmartFusion FPGA (Cortex-M3 Hard IP)
Board: A2F-LNX-EVB Rev 2.A, www.emcraft.com
DRAM:  16 MB
Flash:  8 MB
In:    serial
Out:   serial
Err:   serial
Net:   Core10/100
Hit any key to stop autoboot:  0
## Booting kernel from Legacy Image at 74020000 ...
   Image Name:   Linux-2.6.33-arm1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    821816 Bytes = 802.6 kB
   Load Address: 70008000
   Entry Point:  70008001
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
ï¿½Linux version 2.6.33-arm1 (psl@ocean.emcraft.com) (gcc version 4.4.1 (Sourcery G++ Lite
2010q1-188) ) #1 Fri Dec 06 19:59:56 MSK 2013
CPU: ARMv7-M Processor [411fc231] revision 1 (ARMv7M)
CPU: NO data cache, NO instruction cache
Machine: Actel A2F
...
Freeing init memory: 292K
init started: BusyBox v1.17.0 (2013-12-06 19:58:44 MSK)
~ #
```

The uCLinux kernel is configured to mount a root file system in the external RAM using the initramfs file system. *initramfs* is populated with required files and utilities at the kernel build time and then simply linked into the uCLinux image. *initramfs* doesn't have hard limits on its size and is able to grow using the otherwise unused RAM memory.

The uCLinux image installed on the board provides a demonstration of basic shell and network capabilities of uCLinux Cortex-M. In addition to that, the demo provides support for flash partitioning and persistent data storage using the JFFS2 journalled file system for the external flash memory.

Here is how you can test some of these capabilities.

From a local host, test that the NetFusion is accessible over network (assuming that the NetFusion board is assigned with the 172.17.4.200 IP address):

```
[psl@ocean linux-cortexm-1.12.0]$ ping 172.17.4.200
PING 172.17.4.200 (172.17.4.200) 56(84) bytes of data.
64 bytes from 172.17.4.200: icmp_seq=1 ttl=64 time=3.11 ms
64 bytes from 172.17.4.200: icmp_seq=2 ttl=64 time=0.900 ms
64 bytes from 172.17.4.200: icmp_seq=3 ttl=64 time=0.854 ms
```

On NetFusion, connect to a local host using telnet:

```
~ # telnet 172.17.0.212

Entering character mode
Escape character is '^]'.

Fedora release 12 (Constantine)
Kernel 2.6.32.26-175.fc12.i686 on an i686 (7)
login: psl
Password:
Last login: Mon Jan 31 17:38:57 from 172.17.4.199
[psl@pvr ~]$ logout
Connection closed by foreign host
~ #
```

Start the telnet daemon to allow connections to NetFusion:

```
~ # telnetd
~ #
```

Connect to NetFusion from a local host PC using telnet (hit Enter on the password prompt):

```
[psl@ocean linux-cortexm-1.12.0]$ telnet 172.17.4.200
Trying 172.17.4.200...
Connected to 172.17.4.200.
Escape character is '^]'.

a2f-lnx-evb login: root
Password:
~ #
```

Start the dropbear SSH daemon to allow secure connections to NetFusion:

```
~ # dropbear
~ #
```

Connect to NetFusion from a local host PC using *ssh* (hit Enter on the password prompt):

```
[psl@ocean linux-cortexm-1.12.0]$ ssh root@172.17.4.200
The authenticity of host '172.17.4.200 (172.17.4.200)' can't be established.
DSA key fingerprint is d2:d1:5f:dd:84:65:1d:2f:ee:69:0c:85:d0:22:0c:87.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.17.4.200' (DSA) to the list of known hosts.
root@172.17.4.200's password:
~ #
```

On NetFusion, configure a default gateway and the name resolver. Note how the sample configuration below makes use of the public name server provided by Google. Note also use of "*vi*" to edit target files under uCLinux:

```
~ # route add default gw 172.17.0.1
~ # vi /etc/resolv.conf
nameserver 8.8.8.8
~
```

Use *wget* to download a file from a remote server:

```
~ # wget ftp://ftp.gnu.org/README
Connecting to ftp.gnu.org (140.186.70.20:21)
README               100% |******************************|  1765  --:--:-- ETA
~ # cat README
This is ftp.gnu.org, the FTP server of the the GNU project.

Comments, suggestions, problems and complaints should be reported via
email to <gnu@gnu.org>.
...
```

Use *ntpd* to synchronize the time on the NetFusion with the time provided by a public server:

```
~ # date
Thu Jan  1 00:03:08 UTC 1970
~ # ntpd -p 0.fedora.pool.ntp.org
~ # sleep 5
~ # date
Fri Dec 06 17:06:34 UTC 2013
~ #
```

Mount a directory exported by a development host over NFS:

```
~ # mount -o nolock,rsize=1024 172.17.0.212:/opt/tmp /mnt
~ # mount
rootfs on / type rootfs (rw)
proc on /proc type proc (rw,relatime)
none on /dev/pts type devpts (rw,relatime,mode=600,ptmxmode=000)
172.17.0.212:/opt/tmp on /mnt type nfs
(rw,relatime,vers=3,rsize=1024,wsize=32768,namlen=255,hard,nolock,proto=udp,port=65535,timeo=
7,retrans=3,sec=sys,mountport=65535,mountproto=,addr=172.17.0.212)
~ # ls mnt
CREDITS         busybox         hello.c         hello2.gdb      tests.tgz
address_cache   demo            hello.gdb       runtests.sh     uImage
bacserv         event           hello.good      test            window
bin             hello           hello2          test.log
~ #
```

Start the *HTTP* daemon:

```
~ # httpd -h /httpd/html
~ #
```

From a local host PC, open a Web browser to http://172.17.4.200 and watch the demo web page provided by the Cortex-M NetFusion baseboard.

Next step is to test SNMP. Make sure that the daemon *snmpd* is not running on your host. Create the following configuration file *snmpd.conf*:

```
agentAddress udp:161
rocommunity public  default
rwcommunity private default
```

Then run the daemon *snmpd* on the host with the above configuration:

```
[root@vladimir net-snmp]# snmpd -C -c snmpd.conf -Le -f
```

Then, run the *snmpget* application on the NetFusion and collect information about the host using the *snmpget* application:

```
~ # snmpget -c public 172.17.0.253 .1.3.6.1.2.1.1.1.0
.1.3.6.1.2.1.1.1.0 = STRING: "Linux vladimir.emcraft.com 2.6.32.26-175.fc12.i686.PAE #1 SMP
Wed Aug 24 16:45:50 UTC 2011 i686"
```

Make sure that the daemon *snmptrapd* is not running on your host. Create the following configuration file *snmptrapd.conf*:

```
snmpTrapdAddr udp:162
authCommunity log public
```

Then run the daemon *snmptrapd* on the host with the above configuration:

```
[root@vladimir net-snmp]# snmptrapd -C -c snmptrapd.conf -Le -f
```

Then, send a trap from the NetFusion to the host using the *snmptrap* application:

```
~ # snmptrap -v 1 -c public 172.17.0.253 '1.2.3.4.5.6' '192.193.194.195' 6 99 '55'
1.11.12.13.14.15 s "teststring"
```

On the host PC you should see a log message similar to the one below:

```
2011-08-23 13:45:34 192.193.194.195(via UDP: [172.17.4.201]:44563->[172.17.0.253]) TRAP, SNMP
v1, community public
        iso.2.3.4.5.6 Enterprise Specific Trap (99) Uptime: 0:00:00.55
        iso.11.12.13.14.15 = STRING: "teststring"
```

Erase the third partition of the external flash, format it as a JFFS2 file system and then mount the newly created file system to a local directory. Copy some files to the persistent JFFS2 file system storage:

```
~ # flash_eraseall -j /dev/mtd2
Erasing 64 Kbyte @ 500000 - 100% complete.Cleanmarker written at 4f0000.
~ # mkdir /m
~ # mount -t jffs2 /dev/mtdblock2 /m
~ # cp /bin/busybox /m
~ # cp /etc/rc /m
~ # ls -lt /m
-rwxr-xr-x    1 root     root           389 Jan  1 00:19 rc
-rwxr-xr-x    1 root     root        238724 Jan  1 00:18 busybox
~ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
...
/dev/mtdblock2            5120       436      4684   9% /m
~ # /m/busybox echo Hello from Flash
Hello from Flash
~ # umount /m
```

This is an example *(above)*, but the flash is by default mounted at */flash* on the root system anyway.

Reboot NetFusion:

```
~ # reboot -f
Restarting system.


U-boot 2010.03-a2f-lnx-evb-1.12.0 (Dec 06 2013 - 19:43:45)
...
```

# 4 Software Functionality

## 4.1 Supported Features

The following list summarizes the features and capabilities of uCLinux SmartFusion2, Release 1.12.0:

### 4.1.1 U-boot firmware

- U-boot v2010.03;
- NetFusion initialization from power-on / reset;
- Runs from the internal eNVM and internal SRAM (no external memory required for standalone operation);
- Serial console;
- Ethernet driver for loading images to NetFusion;
- Serial driver for loading images to NetFusion;
- Device driver for built-in flash (eNVM) and self-upgrade capability;
- Device driver for storing environment and uCLinux images in external flash;
- Autoboot feature, allowing boot of OS images from flash or other storage with no operator intervention;
- Persistent environment in flash for customization of NetFusion operation;
- Sophisticated command interface for maintenance and development of NetFusion.

### 4.1.2 uCLinux

- uClinux kernel v2.6.33;
- Boot from compressed and uncompressed images;
- Ability to run critical kernel code from integrated flash of SmartFusion2;
- Serial device driver and uCLinux console;
- Ethernet device driver and networking (*ping, NFS, Telnet, FTP, ntpd*, etc.);
- busybox v1.17;
- POSIX pthreads;
- Process-to-kernel and process-to-process protection using the Memory Protection Unit (MPU) of the SmartFusion2 core;
- Hardened exception handling; an exception triggered by a process affects only the offending process;
- Loadable kernel modules;
- Secure shell (*ssh*) daemon;
- Web server;
- MTD-based flash partitioning and persistent JFFS2 flash file system for external flash;
- Device driver for the USB OTG interface;
- $I^2C$ device driver;
- SPI controller master-mode device driver;
- Device driver for the embedded NVM;
- GPIO device driver.

#### 4.1.3 NetFusion UIO Drivers, Additional Packages

- Default mounting of SPI flash JFFS2 partition to /flash on the file system;
- UIO device drivers for SIP, I2C, UART, GPIO, Audio, LCD, ACI, Temperature and voltage supervision;
- SQLite customized uCLinux build;
- Driver scripts and self-diagnostic shell script;
- Patched core kernel to increase flash partition size and more kernel space for SQLite;

#### 4.1.4 Development tools

- ARMv7-optimized GNU toolchain from CodeSourcery (2010q1) is used for development of U-boot, uCLinux and user-space applications (toolchain must be downloaded separately from the CodeSourcery web site);
- Cross GDB for debugging user-space applications;
- mkimage tool used by the uCLinux kernel build process to create a uCLinux image bootable by U-boot.

#### 4.1.5 Development environment

- uCLinux-hosted cross-development environment;
- Development of multiple projects (embedded applications) from a single installation;
- hello sample project ("Hello, world!" single-process configuration);

### 4.2 Known Problems & Limitations

This section lists known problems and limitations of this release:

- **CONFIG_KERNEL_IN_ENVM** requires disabling **CONFIG_ARM_UNWIND** and **CONFIG_EARLY_PRINTK**.
  *ID*: RT 74683.
  *Workaround*: When enabling **CONFIG_KERNEL_IN_ENVM** in the kernel, disable **CONFIG_ARM_UNWIND** and **CONFIG_EARLY_PRINTK**.
- Debugging of multi-threaded applications using GDB is not supported.
  ID*: RT 77243*. *Workaround*: None. This issue will be resolved in future releases of the product.
- USB hot plug doesn't work on SmarFusion2 under uClinux.
  ID: RT 82415. *Workaround*: Limitation will be fixed in the next release.

# 5 Configuring Starter Kit for uClinux Development

Before you can work with uClinux on the SmartFusion2 NetFusion, you need to have a firmware bootloader called "U-boot" installed in the on-chip non-volatile memory. Your NetFusion comes pre-loaded with U-boot, but it may have been erased if you loaded other demonstration software on your system. Also, to load uCLinux and root filesystem images you will need to set up a *TFTP* server on your PC or in the local network. To load U-boot and to set up a *TFTP* server, please see the steps below.

## 5.1 Powering up and connecting the serial console device

As the NetFusion SOM's Cortex-M3 processor (*embedded inside the SmartFusion2 FPGA*) begins to boot-up, the debug and process can be observed using the USB-UART interface. This is a USB device that your connected PC will recognize as a USB Serial Device and allocate a COM port to it (Windows XP/7/8) **Note:** */dev/ttyS0|1|2 for Linux PCs.*

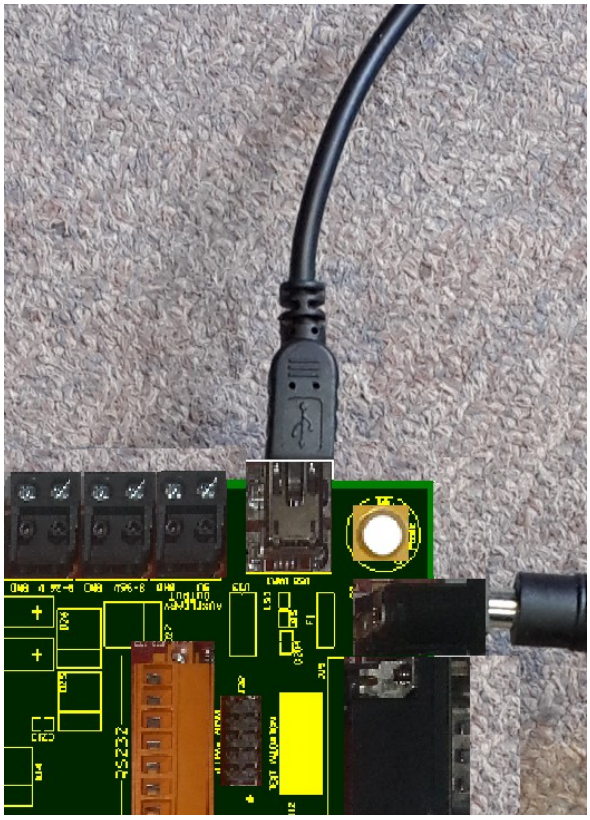


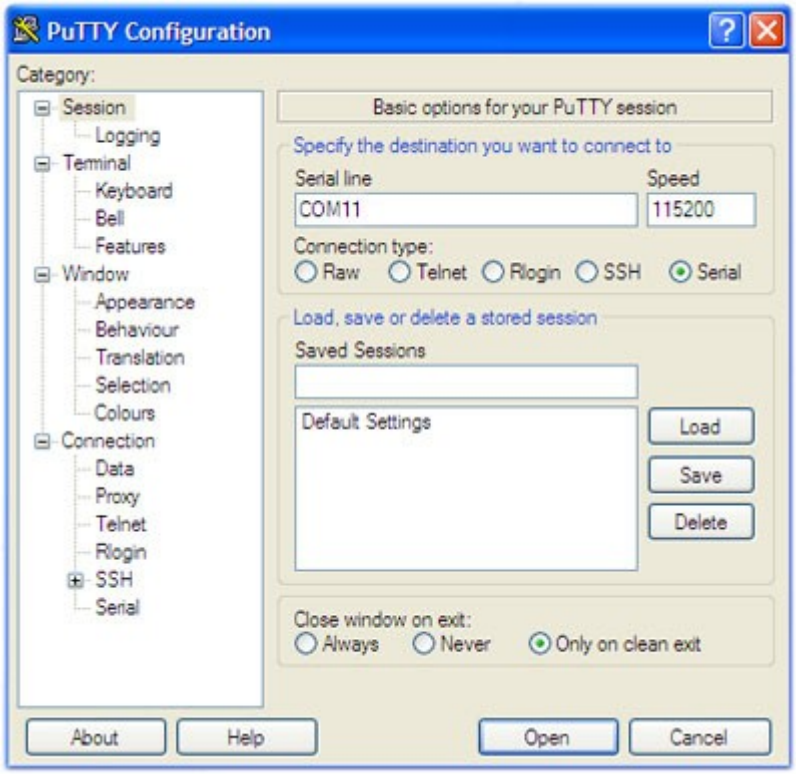
**Figure 2 - USB Cable Connected to the USB-UART Interface from a PC**

Your IDE on the PC should open a window that can see the debug and allow for commands and text to be entered. This will interact with the U-boot.

Once you have programmed the SPI flash with the uCLinux then the interaction and debug will be with the booted uCLinux. This includes how to connect a CAT5 to the 10/100 RJ45 for TFTP network transfer of the uCLinux kernel file.

## 5.2 Connecting a serial console terminal application to the M2S-FG484 SOM NetFusion PCB

- For a Windows development host PC, install the drivers.
- Connect and power up the NetFusion PCB.
- Download and install *PuTTY* (a free serial and networking terminal application).
- Open **Control Panel -> System -> Hardware -> Device Manager**.
- Extend the **Ports (COM & LPT)** item and note the name of the USB serial port. If you do not have the USB serial ports listed, the **drivers** were not probably installed correctly. Please solve this problem before proceeding.
- Open the **PuTTy** window, select **Connection type - Serial**, **Speed - 115200**
- Push **Open**, an empty window will appear:



Push the reset button on the NetFusion. You should see the U-Boot banner:

```
U-Boot 2010.03-linux-cortexm-1.11.0 (July 5 2013 - 19:43:46)

CPU : SmartFusion2 SoC (Cortex-M3 Hard IP)
Freqs: CORTEX-M3=166MHz,PCLK0=83MHz,PCLK1=83MHz
Board: M2S-FG484-SOM Rev 1A, www.emcraft.com
DRAM: 64 MB
In: serial
Out: serial
Err: serial
Net: M2S_MAC
Hit any key to stop autoboot: 0
M2S-FG484-SOM>
```

To verify the serial console is working, please stop at the U-Boot prompt by pushing any key. If needed, hit the reset to restart U-Boot. If there is no activity on the *PuTTY* window, or something else is printed out, then you should install the [.stp] image with U-Boot to the NetFusion M2S-FG484 SOM.

## 5.3 Installing U-boot to the M2S-FG484 SOM of NetFusion

- Install the Libero SoC v11.4.
- Download the [.stp] image from the Nine Ways website nineways.co.uk
- Attach the FlashPro programmer to the development host PC and to the JTAG programming header of NetFusion.
- Start the FlashPro application.
- From the FlashPro IDE, create a new project with an arbitrary name.
- From the main FlashPro window, push **Configure Device**.
- Push **Browse next** to load an existing programming file.
- Browse to the NetFusion project *[.stp]* file and choose it.
- Push **Program** to start programming the M2S-FG484 SOM on the NetFusion with the chosen *[.stp]* image.
- Push the reset button and observe the U-boot banner (*see above*) in the *PuTTY* window.

## 5.4 Installing the TFTP server

- To load uCLinux images to the NetFusion M2S-FG484 SOM, you will need the capabilities of your local network and the *TFTP* server.

- Connect the Ethernet cable to the NetFusion's SOM Ethernet port and to a network switch or to your host PC.

- Install the *TFTP server* and start it. Unblock the application if the Windows firewall asks for permission.

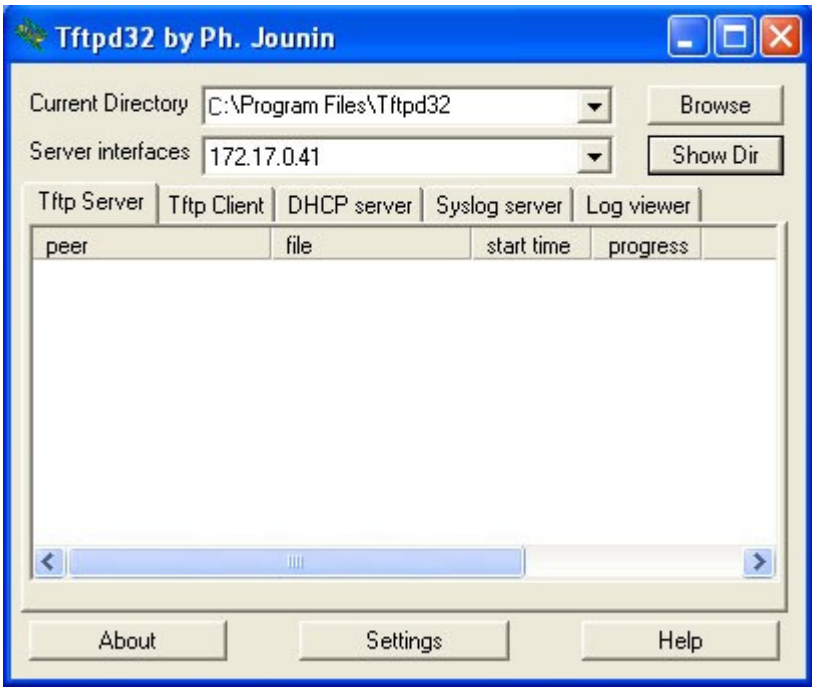



**Figure 3 - TFTP Application on Windows XP/7/8**

- Put the uCLinux image (*netfusion.uImage*) you want to load to the *TFTP* server stage directory (*C:\Program Files\Tftpd32* by default).

## 5.5 Booting the uCLinux Image

To boot the image, you first need to set up networking in U-boot. Generally, you need to set the following U-boot environment variables as in the example below (the actual addresses must match your local network configuration):

```
M2S-FG484 SOM> setenv ipaddr 172.17.4.129
M2S-FG484 SOM> setenv serverip 172.17.0.41
M2S-FG484-SOM> saveenv
```

Check that the connection to the *TFTP* server works:

```
M2S-FG484-SOM> tftp networking.uImage
Link is UP.
Using M2S_MAC device
TFTP from server 172.17.0.41; our IP address is 172.17.4.129
Filename 'networking.uImage'.
Load address: 0xa0007fc0
Loading: ##################################################################
#################################################################
#################################################################
############################
done
Bytes transferred = 3256224 (31afa0 hex)
M2S-FG484-SOM>
```

**Note:** *If your are having trouble with TFTP transfer, please check that the networking setup in U-boot (IP address of the M2S-FG484 NetFusion and the TFTP server, subnet mask) matches your local network settings. If this doesn't help, try to enable the* ***PXE Compatibility*** *checkbox in the Tftpd32* ***Settings*** *menu, TFTP tab.*

Now, you are ready for development in the uCLinux environment and loading of images to the M2S-FG484 NetFusion PCB product.

# 6 SmartFusion2 SOM Board uCLinux Software Set-up

## 6.1 uCLinux Demo and Development Environment

The NetFusion baseboard comes with a preloaded (*uCLinux*) default start project. To initiate the uCLinux bootstrap, simply power-up the NetFusion or reset it by pushing the Reset Push Button.

The uCLinux development environment for NetFusion, including the configuration installed opening the box, can be downloaded from the NetFusion page on the Nine Ways website nineways.co.uk

## 6.2 U-boot Environment

When the SmartFusion2 SOM NetFusion is reset, the uCLinux bootstrap will proceed to boot the U-boot firmware from the on-chip eNVM. Unless interrupted, U-boot proceeds to load the uCLinux bootable image from the SPI flash into the LPDDR and passes control to the uCLinux kernel entry point in LPDDR. uCLinux bootstraps the kernel, mounts a root file system in RAM and enters the interactive shell command interface printing the following output to the serial console:

```
U-boot 2010.03-linux-cortexm-1.12.0 (Dec 06 2013 - 19:43:46)

CPU  : SmartFusion2 SoC (Cortex-M3 Hard IP)
Freqs: CORTEX-M3=166MHz,PCLK0=83MHz,PCLK1=83MHz
Board: M2S-SOM Rev 2A, www.emcraft.com
DRAM:  64 MB
In:    serial
Out:   serial
Err:   serial
Net:   M2S_MAC
Hit any key to stop autoboot:  0
M2S-SOM>
```

U-boot makes use of the so-called environment variables to define various aspects of the system functionality. Parameters defined by the U-boot environment variables include: target IP address, target MAC address, address in RAM where uCLinux bootable images will be loaded, and many more. To examine the current settings of the environment variables, run *printenv* from the U-boot command interface.

U-boot provides a command called *saveenv* that stores the up-to-date run-time environment to the persistent storage, which will be the external flash for the U-boot configuration used on the NetFusion board. You need to call *saveenv* any time when you want to copy current settings of the environment variables to the persistent storage in flash. This is how you can write the current U-boot environment to the external flash:

```
M2S-SOM> saveenv
Saving Environment to SPI Flash...
Erasing SPI flash...Writing to SPI flash...done
M2S-SOM>
```

## 6.3 Ethernet MAC Address

The MAC address of the Ethernet interface is defined by the *ethaddr* U-boot environment variable. The value of the MAC address can be examined from the U-boot command line monitor as follows:

```
M2S-SOM> printenv ethaddr
ethaddr=C0:B1:3C:88:88:88
M2S-SOM>
```

The SmartFusion2 SOM NetFusion board comes with *ethaddr* set to a MAC address uniquely allocated for it. Given that each baseboard has a unique MAC address allocated to it, there is no need to update the *ethaddr* variable (although it is possible to do so).

The MAC address can be changed by modifying the *ethaddr* variable as follows:

```
M2S-SOM> setenv ethaddr C0:B1:3C:88:88:89
```

Don't forget to store your update in the persistent storage using *saveenv* so it is remembered across resets and power cycles.

## 6.4 Network Configuration

You will have to update the network configuration of your NetFusion baseboard to match settings of your local environment.

Typically, all you have to allow loading images over network from a TFTP server is update the U-boot environment variables *ipaddr* (the board IP address) and *serverip* (the IP address of the *TFTP* server).

Update *ipaddr* and *serverip*:

```
M2S-SOM> setenv ipaddr 192.168.0.2
M2S-SOM> setenv serverip 192.168.0.1
```

Then save the updated environment to the external flash so that your changes are persistent across resets/power cycles.

## 6.5 Running Pre-installed uCLinux Image

The NetFusion board comes with uCLinux bootable image for the networking project installed into external flash. To boot this uCLinux configuration onto board just reset the board and let U-boot perform the autoboot sequence.

## 6.6 Loading uCLinux Images

At this point, you are able to load uCLinux bootable images to the board over *TFTP* and either boot them directly or install them to the external flash to allow booting uCLinux from flash on power-up/reset.

On the host, activate the uCLinux SmartFusion2 development environment and build the networking project:

```
-bash-3.2$ . ACTIVATE.sh
-bash-3.2$ cd projects/networking/
-bash-3.2$ make
...
-bash-3.2$
```

Copy the uCLinux bootable image to the *TFTP* download directory:

```
-bash-3.2$ cp netfusion.uImage /tftpboot/vlad/
-bash-3.2$
```

To load the image directly, use the *netboot* U-boot macro:

```
M2S-SOM> setenv image netfusion.uImage
M2S-SOM> run netboot
...
TFTP from server 172.17.0.1; our IP address is 172.17.5.100
Filename 'vlad/networking.uImage'.
...
Loading: #################################################################
         #################################################################
         #############
done
Bytes transferred = 2084704 (1fcf60 hex)
...
   Image Name:   Linux-2.6.33-arm1
   Image Type:   ARM Linux Kernel Image (uncompressed)
...
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

Linux version 2.6.33-arm1 (vlad@ocean.emcraft.com) (gcc version 4.4.1 (Sourcery G++ Lite
2010q1-189) ) #1 Fri Dec 06 15:43:44 MSK 2013
...
```

To load the image into the flash, use the U-boot update macro:

```
M2S-SOM> setenv image netfusion.uImage
M2S-SOM> run update
...
TFTP from server 172.17.0.1; our IP address is 172.17.5.100
Filename 'vlad/networking.uImage'.
...
Loading: #################################################################
         #################################################################
         #############
done
Bytes transferred = 2084704 (1fcf60 hex)
16384 KiB S25FL128S_64K at 0:0 is now current device
Saving Environment to SPI Flash...
Erasing SPI flash...Writing to SPI flash...done
M2S-SOM>
```

Reset the NetFusion baseboard and verify that the newly programmed image boots in the autoboot mode:

```
M2S-SOM> reset
resetting ...

U-boot 2010.03-linux-cortexm-1.12.0 (Dec 06 2013 - 17:19:37)
...
Starting kernel ...
...
init started: BusyBox v1.17.0 (Dec 06 2013 - 17:21:30)
~ #
```

## 6.7 U-boot Build

The BSP distribution comes with U-boot pre-built for the NetFusion board. If however you need to re-build U-boot for your board, please follow the instructions below:

- Install the uCLinux SmartFusion2 distribution to this NetFusion board.
- From the top of the uCLinux SmartFusion2 installation, activate the uCLinux SmartFusion2 cross-compile environment by running *source ACTIVATE.sh*.
- Go to the U-boot source directory (*cd u-boot/).*
- Run the following commands:

```
[psl@pvr u-boot]$ make m2s-som_config
Configuring for m2s-som board...
[psl@pvr u-boot]$ make -s
[psl@pvr u-boot]$ make -s u-boot.hex
```

## 6.8 U-boot Installation

The SmartFusion2 SOM NetFusion arrives with the U-boot firmware pre-installed into the on-chip Flash of the SmartFusion2. The U-boot command line interface provides commands that allow upgrading U-boot on the running target in self-upgrade mode.

However, should you program a faulty U-boot image into SmartFusion2, U-boot can be re-installed using them provided uCLinux SmartFusion2 Libero project and a Microsemi FlashPro tool.

***Note:*** *It is critical that the [.stp] file match the kit version (revision of the SF2 SOM); otherwise, damage to the hardware may result.*

*Here is an example of how this can be done:*

- Start FlashPro on a Windows host PC;
- From the FlashPro IDE, create a new project with an arbitrary name;
- From the main FlashPro window, push Configure Device;
- Push Browse next to load existing programming files. Browse to the uCLinux SmartFusion2 project *[.stp*] file corresponding to your M2S-SOM hardware revision on the NetFusion PCB and choose it.
- Push Program at the top of the main window to program the project onto the SmartFusion2 device and wait for the programming procedure to complete. If the programming completes successfully, a next reset should bring the U-boot start-up messages and the command line interface onto the serial console interface.

# 7 uCLinux Cortex-M Software Development Environment

## 7.1 Distribution and Installation

### 7.1.1 Distribution Image

The uCLinux Cortex-M development software is distributed as a linux-<mcu>-<release>.gzip file available for download from nineways.co.uk

When unpacked, this archive contains, along with the other materials, the software distribution file linux-<MCU>-<release>.tar.bz2. That file can be installed to an arbitrary directory on your Linux development host, as follows:

```
[psl@ocean SF]$ mkdir release
[psl@ocean SF]$ cd release
[psl@ocean release]$ tar xvjf ../linux-A2F-1.12.0.tar.bz2
linux-cortexm-1.12.0/
linux-cortexm-1.12.0/linux/
linux-cortexm-1.12.0/linux/lib/
...
[psl@ocean SF]$ ls -l linux-cortexm-1.12.0
total 24
drwxr-xr-x  3 psl users 4096 2013-12-06 17:06 A2F
-rwxr-xr-x  1 psl users  315 2013-12-06 16:26 ACTIVATE.sh
drwxr-xr-x 24 psl users 4096 2013-12-06 21:32 linux
drwxr-xr-x  5 psl users 4096 2013-12-06 20:16 projects
drwxr-xr-x  3 psl users 4096 2013-12-06 19:14 tools
drwxr-xr-x 31 psl users 4096 2013-12-06 19:16 u-boot
```

You do not need to be the super-user (*root*) in order to install the uCLinux Cortex-M distribution on NetFusion. The installation can be performed from an arbitrary unprivileged user account.

### 7.1.2 Installation Tree

Having been installed onto the Linux host, the uCLinux Cortex-M provides the following files and directories, relative to the top of the installation directory:

- *A2F/* - this is a directory with target components;
- *A2F/busybox/* - busybox source and development tree;
- *A2F/dropbear/* - dropbear *SSH* server source and development tree;
- *A2F/net-snmp/* - source and development tree of the net-snmp package;
- *A2F/uclibc* - source and development tree of the uClibc package;
- *A2F/gdb-2011.03* - source and development tree of the GNU Debugger package;
- *A2F/hostapd-1.0* - source and development tree of the HostAP Daemon package;
- *A2F/wireless_tools* - source and development tree of the Wireless Tools package;
- *A2F/libnl-3.2.11 - source* and development tree of the Netlink Protocol Library package;
- *A2F/netperf-2.6.0 - source* and development tree of the Netperf network benchmarking package;
- *A2F/root* - pre-built target binaries ready for use on the target;
- *u-boot/* - U-boot source and development tree;
- *linux/* - Linux (uClinux) kernel source and development tree;

- *projects/* - contains a Hello World template project and importantly the NetFusion project (embedded applications);
- *tools/* - development tools;
- *tools/bin/mkimage* - utility used by the uCLinux Cortex-M kernel build process to create a bootable U-boot kernel image such as *netfusion.uImage*;
- *ACTIVATE.sh* - shell script you need to perform in order to activate the uCLinux Cortex-M development environment on NetFusion.

### 7.1.3 GNU Cross-Build Tools

As a next step in the uCLinux Cortex-M installation procedure, you need to download the GNU cross-build tools for the Cortex-M target. The tools are available from CodeSourcery and can be downloaded from the following URL:

http://www.codesourcery.com/sgpp/lite/arm/portal/package6503/public/arm-uclinuxeabi/arm-2010q1-189-arm-uclinuxeabi-i686-pc-linux-gnu.tar.bz2

It is recommended that you install the GNU development tools to the tools/ directory of the uCLinux Cortex-M installation, as follows:

```
[psl@pvr linux-cortex-m-1.12.0]$ cd tools/
[psl@pvr tools]$ wget
http://www.codesourcery.com/sgpp/lite/arm/portal/package6503/public/arm-uclinuxeabi/arm-
2010q1-189-arm-uclinuxeabi-i686-pc-linux-gnu.tar.bz2
...
Saving to: arm-2010q1-189-arm-uclinuxeabi-i686-pc-linux-gnu.tar.bz2
...
[psl@pvr tools]$ tar xvfj arm-2010q1-189-arm-uclinuxeabi-i686-pc-linux-gnu.tar.bz2
arm-2010q1/arm-uclinuxeabi/
arm-2010q1/arm-uclinuxeabi/libc/
...
```

It is possible to install the tools to an alternative location, however, should you do that, you will need to modify the *ACTIVATE.sh* script to provide a correct path to the installed tools (specifically, *PATH* must include a correct path to the directory where you have installed the cross-tools).

The tools can be installed from an unprivileged account.

In case you already have the toolchain installed on your development host PC (it would be the case, for instance, if you have more than one installation of uCLinux Cortex-M on your host PC) and the underlying file system supports symbolic links (*DOS FAT* does not), you do not have to install the whole toolchain again. Instead, do the following:

```
[psl@pvr linux-cortexm-1.12.0]$ cd tools/
[psl@pvr tools]$ ln -s <path-to-tools>/arm-2010q1 .
```

***Note:*** *The CodeSourcery tools are 32-bit applications and need a compatibility library to run on 64-bit Linux OS distributions. On the recent Fedora distributions, the compatibility package (e.g. glibc-2.17-14.fc19.i686) is installed by default. On Debian, you need to install it running sudo apt-get install ia32-libs.*

### 7.1.4 Activation

Whenever you want to activate uCLinux Cortex-M development session, go to the top of your uCLinux Cortex-M installation and run:

```
[psl@pvr linux-cortexm-1.12.0]$ source ACTIVATE.sh
```

This command sets up (exports) the following environment variables required by the uCLinux Cortex-M development environment:

**INSTALL_ROOT**=<*dir*> - root directory of the installation. This variable is used by the uCLinux Cortex-M make system as well as in the *netfusion.initramfs* file to provide a reference to the uCLinux Cortex-M installation directory;

**CROSS_COMPILE**=*arm-uclinuxeabi-* - Reference to the cross-tools used to build U-boot and the uCLinux kernel;

**CROSS_COMPILE_APPS**=*arm-uclinuxeabi-* - Reference to the cross-tools used to build user-space applications and tools;

**PATH**=*$INSTALL_ROOT/tools/arm-2010q1/bin:$INSTALL_ROOT/tools/bin:*<orig_path> - Path to the cross-tools used by the shell;

**MCU**=<*arch*> - MCU architecture supported by the uCLinux Cortex-M distribution.

### 7.1.5 Dependency on NetFusion Components

The uCLinux Cortex-M distribution has the following dependencies on Linux-host PC software components. Please consult the notes below in case the uCLinux Cortex-M development environment does not function as documented in the sections that follow.

- The U-boot, *busybox* and uCLinux kernel build systems require that certain host packages be installed on the development host to function correctly. These packages are*: make, gcc, perl* and some others. Please refer to *linux/Documentation/Changes* for a list of host tools required to build the uCLinux kernel. The same set of tools is required for the U-boot and *busybox* build.
- The uCLinux Cortex-M GNU debugger (*tools/bin/arm-uclinuxeabi-gdb*) and the configuration scripts require that the following dynamically-linked libraries to be installed on the host:
  - libncurses.so.5 and libtinfo.so.5 (the *ncurses-libs* package);
  - libexpat.so.1 (the *expat* package).
- The CodeSourcery cross-build tools. Please refer to the toolchain README at http://www.codesourcery.com/sgpp/lite/arm/portal/doc7632/getting-started.pdf.
- The *tools/mkimage* utility. It is expected that the utility will work as is on most of today's uCLinux distributions. If it fails to run on a certain host, you can build it for your specific uCLinux host from the sources included in the uCLinux Cortex-M distribution by running make tools from the top of the U-boot tree.
- To exercise the sample projects included to the uCLinux Cortex-M distribution, you need to install the following host packages:
  - *NFS* server;
  - *TFTP* server;
  - Terminal emulator (*picocom, minicom, kermit or PuTTY*).

With NetFusion using the USB for the serial console connection you might need to install the FTDI USB-UART driver from http://www.ftdichip.com/Drivers/VCP.htm for your host PC.

## 7.2 Projects Framework

### 7.2.1 Multiple Target Projects

In the NetFusion uCLinux Cortex-M installation, there is a directory called projects, which provides a framework that you will be able to use to develop multiple projects (embedded applications) from a single installation of uCLinux Cortex-M. This directory has the following structure:

- *projects/Rules.make* - build rules common for all projects;
- *project1/* - *Project1* source files and build tree;
- *project2/* - *Project2* source files and build tree;
- ...

The installation provides two sample projects, called *hello* (A Hello, world! application) AND *netfusion* (the main NetFusion start project). You will be able to add more projects to this directory and develop your own embedded applications using this framework.

Each project directory (such as *projects/hello*) contains the following configuration files:

- *project*.kernel.${MCU} - Kernel configuration for this project. Note that the kernel configuration is specific for the microcontroller architecture supported by the distribution;
- project.*busybox* - busybox configuration for this project;
- project.*initramfs* - File defining content of the initramfs filesystem for this project.

The main NetFusion directory (that is *projects/netfusion*) contains the following configuration files:

- *netfusion*.kernel.${MCU} - Kernel configuration for NetFusion.
- *netfusion*.busybox - busybox configuration for NetFusion.
- netfusion.*initramfs* - File defining content of the initramfs file system for NetFusion.

When you run make linux (or simply *make*) from the NetFusion project directory, the build system builds project-specific versions of the uCLinux kernel and *busybox*, then creates an *initramfs* filesystem containing the newly built *busybox* binary as well as other target files defined by the *initramfs* filesystem specification file, and finally wraps it all up into a uCLinux image *(netfusion.uImage)* ready for download to the target by U-boot.

Each project directory has a *Makefile* identifying the following make variables. You need to set these variables correctly for your project:

- SAMPLE - the name of the project. The downloadable uCLinux image has this name, with the *[.uImage]* extension. Additionally, SAMPLE is passed to the *initramfs* build subsystem as a reference to the directory where file system binaries reside.
- CUSTOM_APPS - this variable provides a list of the sub-directories containing custom applications for the project. Custom applications are built in the specified order, prior to building the uCLinux kernel and *initramfs* images. If a project doesn't have custom applications, the variable should be left empty.

The following make targets are implemented in *projects/Rules.make* common build rules file:

- *all* or *linux* - build an uncompressed uCLinux image ready for download to NetFusion and place it to the project directory as for instance *netfusion*.uImage;
- *kclean* - clean up the uCLinux kernel source tree by running *make clean in $(INSTALL_ROOT)/linux;*

- *bclean* - clean up the *busybox* source tree by running make clean in *$(INSTALL_ROOT)/A2F/busybox*;
- *aclean* - clean up the custom application source trees by running make clean in each application source directory, as listed in CUSTOM_APPS;
- *clean* - clean up the entire project, by removing for instance *netfusion.uImage* and then cleaning up the uCLinux kernel and *busybox* trees, as described above. Additionally, if CUSTOM_APPS is not empty, make clean is performed in each custom application sub-directory;
- *kmenuconfig* - configure the uCLinux kernel by running make *menuconfig* in the uCLinux source tree. Copy the resultant configuration file to the project directory as *netfusion*.kernel.${MCU};
- bmenuconfig - configure the *busybox* tool by running make *menuconfig* in the *busybox* source tree. Copy the resultant configuration file to the project directory as *netfusion*.busybox;
- *clone new*=newproject - clone the current project into a new project with a specified name. The current project directory (with all sub-directories) is copied into the new project directory. The project kernel, *busybox* and initramfs configuration files are copied with the new name. In Makefile SAMPLE is set to the name of the new project.

The main idea behind this framework is that each project keeps its own configuration for the uCLinux kernel (in the *netfusion*.kernel.${MCU} file), its own definition of the contents of the NetFusion file system (in the *netfusion*.initramfs file) and its own configuration of *busybox* (in the *netfusion.busybox* file). These files are enough to rebuild both the project kernel and the project file system (with a specific configuration of the *busybox* tool embedded) from scratch.

If a project makes use of a custom application specific to the project, such an application must be built from the sources located in an arbitrarily-named sub-directory local to the project directory. There could be several sub-directories in a project, one per a custom application. Each subdirectory will have its own Makefile defining rules for building the custom application from the corresponding sources.

Each custom application listed in CUSTOM_APPS must have a Makefile in the custom application sub-directory defining the following makefile targets:

- *all* - build the application from the sources;
- *clean* - clean anything that has been previously built.

Content of the initramfs file system is defined for a project in the file named *netfusion.initramfs*. Note that this file makes use of ${INSTALL_ROOT} to provide a reference to the top of the uCLinux Cortex-M installation directory.

### 7.2.2 Hello World Demo Project

This additional directory resides in *projects/hello*. It is the only additional project other than the default main NetFusion project.

A single-application uCLinux configuration demonstrating the minimal possible kernel/initramfs memory requirements. The application starts as an init process as soon as the kernel has mounted the root filesystem, mounts the /proc filesystem and prints out the content of */proc/meminfo*. Then it proceeds to print a "Hello" message in an endless loop. There is no way to interrupt or stop the application (rather than resetting the system).

# 8 U-boot Loader Facility

## 8.1 Pointers to Detailed Information on U-boot

uCLinux Cortex-M makes use of the U-boot firmware to bring the Cortex-M target up from a power-on / reset and to boot uCLinux onto the NetFusion.

The U-boot is a popular firmware monitor developed and maintained by DENX Software Engineering (www.denx.de). DENX publishes extensive U-boot user documentation at their web site.

This section provides a brief introduction to U-boot as relevant to the uCLinux Cortex-M software environment.

## 8.2 U-boot: Theory of Operation

A typical uCLinux Cortex-M boot-up sequence is as follows:

- U-boot firmware runs on the NetFusion from the integrated flash (eNVM) / eSRAM (no external memory required) and performs all required initialization from power-on / reset, including setting up the memory controller for external memory accesses;
- U-boot relocates uCLinux image from external flash to external RAM and passes control to the kernel entry point in RAM. External flash resides at the memory interface. uCLinux image can be compressed to save the flash storage; in this case, U-boot explodes the image when relocating it to RAM;
- uCLinux proceeds to boot up and mount a RAM-based filesystem (*initramfs*) as a root filesystem. *initramfs* is populated with required files and directories at build time and is then simply linked into the kernel.

U-boot operation is controlled at run-time by the so-called "environment variables", which are stored in persistent storage such as external flash. Each environment variable controls a certain aspect of U-boot operation. For instance, there is an environment variable called *bootargs* that defines a kernel command string that U-boot passes to the uCLinux kernel as the parameters.

## 8.3 Selected Use Scenarios

### 8.3.1 U-boot Environment

To manipulate the U-boot environment the following commands are used:

- *printenv* <*var*> - print the value of the variable *var.* Without arguments, print all environment variables:

```
A2F-LNX-EVB> printenv
bootargs=console=ttyS0,115200 panic=10
bootcmd=run flashboot
bootdelay=3
baudrate=115200
hostname=a2f-lnx-evb
loadaddr=70000000
addip=setenv bootargs ${bootargs}
ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:eth0:off
flashaddr=74020000
flashboot=run addip;bootm ${flashaddr}
netboot=tftp ${image};run addip;bootm
image=blabla.uImage
ethact=Core10/100
stdin=serial
stdout=serial
stderr=serial
ethaddr=00:22:44:66:88:AA

Environment size: 430/4092 bytes
A2F-LNX-EVB>
```

- *setenv* <*var*> <*val*> - set the variable *var* to the value val:

```
A2F-LNX-EVB> setenv image netfusion.uImage
A2F-LNX-EVB> printenv image
image=netfusion.uImage
A2F-LNX-EVB>
```

Running *setenv* <*var*> will un-set the U-boot variable.

- *saveenv* - save the up-to-date U-boot environment, possibly updated, using *setenv* commands, into persistent storage:

```
A2F-LNX-EVB> saveenv
Saving Environment to Flash...
Un-Protected 2 sectors
Erasing Flash...
.. done
Erased 2 sectors
Writing to Flash... done
Protected 2 sectors
A2F-LNX-EVB>
```

### 8.3.2 Autoboot

The autoboot sequence in U-boot is controlled by the following environment variables:

- *bootcmd* - command to execute automatically after reset. To disable the autoboot, undefine this variable;
- *bootdelay* - delay, in seconds, before running the *autoboot* command. During the *bootdelay* countdown, you can interrupt the *autobooting* by pressing any key.

### 8.3.3 Networking in U-boot

U-boot supports downloading of images over network using the *TFTP* service.

To enable networking, the following U-boot variables must be set to values matching the local network configuration:

- *ipaddr* - IP address of your NetFusion SOM board;
- *serverip* - IP address of the *TFTP* server;
- *netmask* - local network mask (optional variable);
- *image* - filename of an image loaded from the *TFTP* server (optional variable).

After you have defined the above variables, you can use the *tftp* U-boot command to download images to RAM. For example:

```
A2F-LNX-EVB> tftp netfusion.uImage
Core10/100: link up (100/Full)
Using Core10/100 device
TFTP from server 172.17.0.1; our IP address is 172.17.10.200
Filename 'netfusion.uImage'.
Load address: 0x70000000
Loading: ##########################################################
done
Bytes transferred = 821888 (c8a80 hex)
A2F-LNX-EVB>
```

### 8.3.4 Flash-related Commands

To program uCLinux image to the flash, use the update U-boot macro as follows:

```
A2F-LNX-EVB> print update
update=tftp ${image};prot off ${flashaddr} +${filesize};era ${flashaddr} +${filesize};cp.b
${loadaddr} ${flashaddr} ${filesize}
A2F-LNX-EVB> setenv image netfusion.uImage
A2F-LNX-EVB> run update
```

Alternatively, you can program uCLinux image to the flash using the following sequence of commands (the commands are only valid for the NOR type of the flash:

- Download the image to RAM:

```
A2F-LNX-EVB> tftp netfusion.uImage
Core10/100: link up (100/Full)
Using Core10/100 device
TFTP from server 172.17.0.1; our IP address is 172.17.10.200
Filename 'netfusion.uImage'.
Load address: 0x70000000
Loading: ###########################################################
done
Bytes transferred = 821888 (c8a80 hex)
A2F-LNX-EVB>
```

- Unprotect flash:

```
A2F-LNX-EVB> protect off all
Un-Protect Flash Bank # 1
.....................................................................................
..................................... done
A2F-LNX-EVB>
```

- Erase the destination flash area:

```
A2F-LNX-EVB> erase ${flashaddr} +${filesize}

.............. done
Erased 13 sectors
A2F-LNX-EVB>
```

- Copy the image from RAM to flash:

```
A2F-LNX-EVB> cp.b ${loadaddr} ${flashaddr} ${filesize}
Copy to Flash... done
A2F-LNX-EVB>
```

### 8.3.5 U-boot Self-Upgrade

The uCLinux Cortex-M distribution provides a full tree of the U-boot source files. This allows you to configure or otherwise enhance U-boot for your specific needs.

To build U-boot, do the following from the uCLinux Cortex-M installation on NetFusion (refer to Section 7):

- Go to the uCLinux Cortex-M installation and activate it (unless you have activated it already):

```
[psl@ocean linux-cortexm-1.12.0]$ source ACTIVATE.sh
[psl@ocean linux-cortexm-1.12.0]$
```

- Go to the top of the U-boot source tree:

```
[psl@ocean linux-cortexm-1.12.0]$ cd u-boot/
[psl@ocean u-boot]$
```

- Configure U-boot for your NetFusion board. Please refer to the Starter Kit Guide (or BSP Guide) for your hardware board for the name of the corresponding U-boot configuration target. For example, the following command configures U-boot for the NetFusion board:

```
[psl@ocean u-boot]$ make a2f-lnx-evb_config
Configuring for a2f-lnx-evb board...
[psl@ocean u-boot]$
```

- Build the U-boot image ready for download to the NetFusion over *TFTP* and copy it to the *TFTP* server directory:

```
[psl@ocean u-boot]$ make
[psl@ocean u-boot]$ cp u-boot.bin /tftpboot/psl/a2f
```

To upgrade the firmware, do the following on the NetFusion:

- Configure the networking in U-boot if necessary.
- Download the U-boot image from the *TFTP* server:

```
A2F-LNX-EVB> tftp psl/a2f/u-boot.bin
Core10/100: link up (100/Full)
Using Core10/100 device
TFTP from server 172.17.0.1; our IP address is 172.17.10.200
Filename 'psl/a2f/u-boot.bin'.
Load address: 0x70000000
Loading: #######
done
Bytes transferred = 96052 (17734 hex)
A2F-LNX-EVB>
```

Program the new U-boot image to the embedded flash of the MCU using the *cptf* command. Note that the *cptf* command automatically resets the NetFusion upon completion of the eVNM upgrade, which calls the newly programmed image from the reset vector in the eNVM. Note also that the first parameter passed to the *cptf* command specifies the base address of the embedded flash area being updated by the command. Depending on your CPU, pass the following address as the first parameter to *cptf*:

- STM32F = 0x08000000
- LPC17xx = 0x0
- SmartFusion and SmartFusion2 = 0x0
- Kinetis = 0x0

As an example, the following command performs the U-boot self-upgrade on NetFusion:

```
A2F-LNX-EVB> cptf 0x0 ${loadaddr} ${filesize} 1
cptf: Updating eNVM. Please wait ...


U-boot 2010.03 (Dec 06 2013 - 20:15:52)

CPU: SmartFusion FPGA (Cortex-M3 Hard IP)
Board: A2F-LNX-EVB Rev 2.A, www.emcraft.com
...
A2F-LNX-EVB>
```

***Note:*** *be extra-careful when performing the cptf command specified above. In case you program an incorrect U-boot image to the eNVM, this will render the board non-bootable. The only resort in this scenario is to program the U-boot image to the board over the JTAG port.*

# 9 Sample Development Session

## 9.1 Sample Session Outline

This sample session illustrates a possible software development cycle in uCLinux Cortex-M.

We will do the following:

- Create a new project by cloning it off of the developer demo project.
- Modify the new project to automatically NFS-mount a development tree from the uCLinux host where we have installed uCLinux Cortex-M.
- Program the new project image into flash so that it *autoboots* on NetFusion on each power-up / reset.
- Develop a new custom application and a kernel module and debug them from the NFS-mounted host directory, without having to even reboot the NetFusion target.

## 9.2 Create a New Project

This section creates a new project as a clone of the existing project and makes sure the new project builds:

Start off of the developer project and create a clone called *my_developer*:

```
-bash-3.2$ pwd
/home/vlad/test/linux-cortexm-1.12.0/projects/developer
-bash-3.2$ make clone new=my_developer
New project created in /home/vlad/test/linux-cortexm-1.12.0/projects/my_developer
-bash-3.2$
```

Go to the new project directory, build it and copy the downloadable uCLinux image to the *TFTP* server directory:

```
-bash-3.2$ cd ../my_developer/
-bash-3.2$ make
...
  Image arch/arm/boot/uImage is ready
...
-bash-3.2$ cp my_developer.uImage /tftpboot/vlad/v
```

## 9.3 Set Up Target for Debugging of the New Project

This section sets up U-boot for debugging of the new project on the NetFusion:

Reset the NetFusion and enter the U-boot command monitor, hitting any key to stop the autoboot:

```
...
Hit any key to stop autoboot:  0
A2F-LNX-EVB>
```

Define the IP addresses for the NetFusion and a *TFTP* server; define the name of the image to be downloaded by *tftpboot*:

```
A2F-LNX-EVB> setenv ipaddr 172.17.4.150
A2F-LNX-EVB> setenv serverip 172.17.0.1
A2F-LNX-EVB> setenv image vlad/v
```

Check that U-boot defines appropriate commands (macros) for booting the uCLinux image from a *TFTP* server and running it on the NetFusion:

```
A2F-LNX-EVB> printenv netboot
netboot=tftp ${image};run addip;bootm
A2F-LNX-EVB> printenv addip
addip=setenv bootargs ${bootargs}
ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:eth0:off
```

Save the updated environment in flash:

```
A2F-LNX-EVB> saveenv
Saving Environment to Flash...
Un-Protected 2 sectors
Erasing Flash...
.. done
Erased 2 sectors
Writing to Flash... done
Protected 2 sectors
A2F-LNX-EVB>
```

Boot the uCLinux image over the network and test that the networking is functional. Note that given a correct *ipaddr* setting in U-boot, the kernel brings up the Ethernet interface in uCLinux automatically, without an explicit *ifconfig* command:

```
A2F-LNX-EVB> run netboot
Core10/100: link up (100/Full)
Using Core10/100 device
TFTP from server 172.17.0.1; our IP address is 172.17.4.150
Filename 'vlad/v'.
Load address: 0x70000000
Loading: #################################################################
         #################################
done
Bytes transferred = 1439232 (15f600 hex)
## Booting kernel from Legacy Image at 70000000 ...
   Image Name:   Linux-2.6.33-arm1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1439168 Bytes =  1.4 MB
   Load Address: 70008000
   Entry Point:  70008001
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

ÐªLinux version 2.6.33-arm1 (vlad@ocean.emcraft.com) (gcc version 4.4.1 (Sourcery G++ Lite
2010q1-188) ) #8 Fri Mar 25 17:26:37 MSK 2011
...
IP-Config: Complete:
     device=eth0, addr=172.17.4.150, mask=255.255.0.0, gw=255.255.255.255,
     host=a2f-lnx-evb, domain=, nis-domain=(none),
     bootserver=172.17.0.1, rootserver=172.17.0.1, rootpath=
Freeing init memory: 328K
init started: BusyBox v1.17.0 (2013-12-06 19:58:44 MSK)

~ # ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1): 56 data bytes
64 bytes from 172.17.0.1: seq=0 ttl=64 time=11.617 ms
64 bytes from 172.17.0.1: seq=1 ttl=64 time=2.016 ms
...
```

## 9.4 Update the New Project

This section updates the new project for the required functionality and validates it on NetFusion:

In the new project, update the NetFusion start-up script so that it automatically mounts the projects directory from the uCLinux Cortex-M installation on the NetFusion. This makes all projects immediately available on the NetFusion allowing you to edit, build and test your sample applications and loadable device drivers without having to re-flash or even reboot NetFusion:

```
-bash-3.2$ vi local/rc
#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devpts none /dev/pts
mkdir /mnt
mount -o nolock,rsize=1024 172.17.0.1:/home/vlad/test/linux-cortexm-1.12.0/projects /mnt
ifconfig lo 127.0.0.1
```

Build the updated project and copy it to the *TFTP* server directory:

```
-bash-3.2$ make; cp my_developer.uImage /tftpboot/vlad/v
...
```

Reboot the NetFusion, load the updated image over the network and test it:

```
~ # reboot -f
Restarting system.

...
Hit any key to stop autoboot:  0
A2F-LNX-EVB> run netboot
...
init started: BusyBox v1.17.0 (2013-12-06 19:58:44 MSK)
~ # mount
rootfs on / type rootfs (rw)
proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
none on /dev/pts type devpts (rw,relatime,mode=600,ptmxmode=000)
172.17.0.1:/home/vlad/test/linux-cortexm-1.12.0/projects on /mnt type nfs
(rw,relatime,vers=3,rsize=1024,wsize=32768,namlen=255,hard,nolock,proto=udp,port=65535,timeo=
7,retrans=3,sec=sys,mountport=65535,mountproto=,addr=172.17.0.1)
~ # ls -lt /mnt
drwxr-xr-x    4 19270    19270         4096 Mar 25  2011 my_developer
drwxr-xr-x    4 19270    19270         4096 Mar 25  2011 developer
-rw-r--r--    1 19270    19270         3581 Mar 25  2011 Rules.make
drwxr-xr-x    3 19270    19270         4096 Mar 25  2011 hello
drwxr-xr-x    3 19270    19270         4096 Mar 25  2011 networking
~ #
```

## 9.5 Install the New Project to Flash

This section installs the new project to flash so that it automatically boots up on NetFusion on any power-up / reset:

At the U-boot prompt, load the uCLinux image into the target RAM over *TFTP* and program it to flash:

```
A2F-LNX-EVB> print image
image=vlad/v
A2F-LNX-EVB> run update
Core10/100: link up (100/Full)
Using Core10/100 device
TFTP from server 172.17.0.1; our IP address is 172.17.4.150
Filename 'vlad/v'.
Load address: 0x70000000
Loading: ##################################################################
         ##################################
done
Bytes transferred = 1439232 (15f600 hex)
Un-Protect Flash Bank # 1
.....................................................................................
..................................... done
Erased 22 sectors
Copy to Flash... done
A2F-LNX-EVB>
```

Reset the NetFusion baseboard and make sure that the new project boots from flash in the autoboot mode:

```
A2F-LNX-EVB> reset
resetting ...
...
Hit any key to stop autoboot:  0
## Booting kernel from Legacy Image at 74020000 ...
...
init started: BusyBox v1.17.0 (2013-12-06 19:58:44 MSK)
~ # ls -lt /mnt
drwxr-xr-x    4 19270    19270         4096 Mar 25  2011 my_developer
...
~ #
```

## 9.6 Develop a Custom Loadable Device Driver over NFS

Go to the application sub-directory in your project's directory:

```
-bash-3.2$ cd app/
-bash-3.2$ pwd
/home/vlad/test/linux-cortexm-1.12.0/projects/hello/app
```

The developer project you have cloned your new project off already provides a loadable kernel device driver implemented in *sample.c*. The device driver allows reading "device data" from a pseudo-device, which keeps its data in a character array defined in *sample.c*. Let's enhance the device driver to allow changing the "device data" by writing into the character array. Note that, this being a simple example of a development session, the code below assumes that the user-supplied "device-data" does not exceed 1023 bytes.

```
-bash-3.2$ vi sample.c
...
/*
 * Device "data"
 */
static char sample_str[1024] = "This is the simplest loadable kernel module\n";
...
/*
 * Device write
 */
static ssize_t sample_write(struct file *filp, const char *buffer,
                            size_t length, loff_t * offset)
{
        int ret = 0;

        /*
         * Check that the user has supplied a valid buffer
         */
        if (! access_ok(0, buffer, length)) {
                ret = -EINVAL;
                goto Done;
        }

        /*
         * Write the user-supplied string into the sample "device string".
         */
        strncpy(sample_str, buffer, length);
        sample_str[length] = '\0';
        *offset += length + 1;
        ret = length;

Done:
        d_printk(3, "length=%d\n", length);
        return ret;
}
```

Build the updated device driver:

```
-bash-3.2$ make
```

On NetFusion, go to the application directory:

```
/mnt/my_developer/app # cd /mnt/hello/app/
```

Install the updated device driver:

```
/mnt/hello/app # insmod sample.ko
```

Test that the read operation returns the content of the built-in "device data":

```
/mnt/hello/app # cat /dev/sample
This is the simplest loadable kernel module
```

Write into /dev/sample in order to change the "device data" and test the device returns the updated data on a next read:

```
/mnt/hello/app # cat > /dev/sample
This is the new content of /dev/sample
^D
/mnt/hello/app # cat /dev/sample
This is the new content of /dev/sample
```

Unload the device driver:

```
/mnt/hello/app # rmmod sample
```

Iterate to update and test your custom device driver from the NFS-mounted host directory.

## 9.7 Develop a Custom User-Space Application over NFS

You are in the application sub-directory in your project's directory:

```
-bash-3.2$ pwd
/home/vlad/test/linux-cortexm-1.12.0/projects/hello/app
```

The developer project you have cloned your new project off already provides a user-space application implemented in *app.c*. This is an example so let's add a simple print-out to the application code:

```
-bash-3.2$ vi app.c
...
        printf("%s: THIS IS CONTENT OF %s:\n", app_name, dev_name);

        /*
         * Read the sample device byte-by-byte
         */
        while (1) {
...
```

Build the updated application:

```
-bash-3.2$ make
```

On NetFusion, install the device driver and test the updated application:

```
/mnt/hello/app # insmod sample.ko
/mnt/hello/app # ./app
./app: THIS IS CONTENT OF /dev/sample:
This is the simplest loadable kernel module
```

Unload the device driver:

```
/mnt/hello/app # rmmod sample
```

Iterate to update and test your custom application from the NFS-mounted host directory.

# 10 Building uClibc

The NetFusion uCLinux distribution provides the uClibc library prebuilt for the Cortex-M architecture. You can just link your applications with the provided library and never worry about having to build the library from scratch. In the installation tree, the library is available in the following directory relative to the top of the installation:

```
-bash-3.2$ pwd
/home/vlad/test/linux-cortexm-1.12.1
-bash-3.2$ cd A2F/uclibc/
-bash-3.2$
```

In some situations however, our customers asks us how to rebuild the library. Typically, they want to do that in order to enable some additional functionality available from the library but not included in the prebuilt configuration.

This application note explains how to build the uClibc library.
First step is to activate the cross development environment on the host PC. This is done by running the *ACTIVATE.sh* script from the top of the installation tree:

```
-bash-3.2$ pwd
/home/vlad/test/linux-cortexm-1.12.1
-bash-3.2$ source ACTIVATE.sh
-bash-3.2$
```

Go to the uClibc directory:

```
-bash-3.2$ cd A2F/uclibc/
-bash-3.2$
```

Copy the default configuration file to the working *config* that will be used by the build procedure:

```
-bash-3.2$ cp dot_config_pthreads .config
-bash-3.2$
```

Edit the *config* file to provide the correct path to the header files. To do that, search for **KRNL_HDR** and replace it with *../root/usr/include/:*

```
-bash-3.2$ vi .config
...
# UCLIBC_HAS_FENV is not set
KERNEL_HEADERS="../root/usr/include/"
UCLIBC_UCLINUX_BROKEN_MUNMAP=y
...
```

Proceed to update the library configuration file to your liking. The example below enables the RPC related build-time options in the configuration:

```
...
# UCLIBC_HAS_IPV6 is not set
UCLIBC_HAS_RPC=y
UCLIBC_HAS_FULL_RPC=y
UCLIBC_HAS_REENTRANT_RPC=y
# UCLIBC_USE_NETLINK is not set
...
```

Remember to save your edits when exiting the editor.

Build the internal configuration file that will be used to by the library build procedure:

```
-bash-3.2$ make -s ARCH_CFLAGS= 'CPU_CFLAGS=-mthumb -march=armv7 -mfix-cortex-m3-ldrd'
oldconfig
#
# configuration written to ./.config
#
-bash-3.2$
```

Build the updated library. Ignore the compiler warnings:

```
-bash-3.2$ make ARCH_CFLAGS= 'CPU_CFLAGS=-mthumb -march=armv7 -mfix-cortex-m3-ldrd' -j2 -s
...
-bash-3.2$
```

Install the newly build library wherever you would like to install it. You can of course install it to the same directory where it is installed in the uCLinux distribution, as shown in the below command:

```
-bash-3.2$ make ARCH_CFLAGS= 'CPU_CFLAGS=-mthumb -march=armv7 -mfix-cortex-m3-ldrd' -s
PREFIX=$INSTALL_ROOT/A2F/root install
-bash-3.2$
```

If you would like to preserve the original library and install the updated library to some other directory, specify the target directory with **PREFIX** in the above command. Whenever you have installed the updated library, provide the correct path to it with the -L and -I compiler flags when building your application. For instance, for the installation command above, you need to specify the following when building your application:

```
-bash-3.2$ arm-uclinuxeabi-gcc -o test test.c -mcpu=cortex-m3 -mthumb -
I$INSTALL_ROOT/A2F/root/usr/include -L$INSTALL_ROOT/A2F/root/usr/lib
-bash-3.2$
```

# 11 Building Multi-Threaded Applications

The version of the uClibc included with the CodeSourcery tool chain does not provide support for POSIX threads. To address this limitation, Emcraft has enabled POSIX threads in uClibc and included the updated library built for Thumb in our uClinux distribution.

In order to build a multi-threaded application and link it with the appropriate uClibc binary, do the following. After activation of the cross-build environment (*source ACTIVATE.sh*), run the following command (*linux-dp.c* being the application in the below example):

```
arm-uclinuxeabi-gcc -o linux-dp linux-dp.c -I \
${INSTALL_ROOT}/A2F/root/usr/include -mcpu=cortex-m3 -mthumb -L \
${INSTALL_ROOT}/A2F/root/usr/lib -pthread
```

# 12 NetFusion Project in the uCLinux Tree

## 12.1 What is the NetFusion project with-in the uCLinux?

The NetFusion baseboard is in effect, an add-on to the M2S050-F484-SOM as opposed to the other way round. Consider the brain-board SOM has having the NetFusion PCB hosting it and therefore, what is statutorily available for the Emcraft M2S-F484-SOM in terms of the uCLinux development environment, will required “add-on” software/drivers etc for the NetFusion baseboard. It is easy to get confused and incorrectly assume that because the M2S-F484-SOM brain-board (containing the SmartFusion2 FPGA and the ARM Cortex-M3 processor with-in) that is a sub-set component of the NetFusion PCB is an extension of the baseboard. Software wise, it is completely the opposite.

The SOM plug-in brain-board hosts the processor and therefore is the seat of the OS and the surrounding sub-system environment. This means that the uCLinux development directory structure on the host cross-compile PC is the primary install which contains sample and template application projects with-in. As NetFusion is effectively an attachment to the M2S-F484-SOM uCLinux project, it is logical for the NetFusion hardware and functionality to be an inner “application” project in the uCLinux environment.

Therefore, the *projects/netfusion* directory exists in the required build as a sub-directory which contains the user-space device drivers (UIO) and included and bundled useful applications and packages. The NetFusion build becomes the primary makefile build and this refers on to the main uCLinux kernel when required.

The installation of the entire NetFusion/Emcraft development on the host Linux PC for cross-compilation is derived from just three simple zip file and installation steps. Firstly, you download and unpack the Emcraft uCLinux development tree. Secondly, download and unpack the cross-compile ARM Cortex-M3 toolset which installs into a sub-directory of the first installation (*above*). Finally, download and unpack the NetFusion project into the *projects/* directory deepest with-in the Emcraft uCLinux tree also.

The host PC has to be Linux natively and is ideal on Ubuntu and similar. No prior installation dependencies exist and as long as the package three files aforementioned can be fetched and installed with *tar* or *gzip*, the process is simple and you can be up and running very quickly and easily.

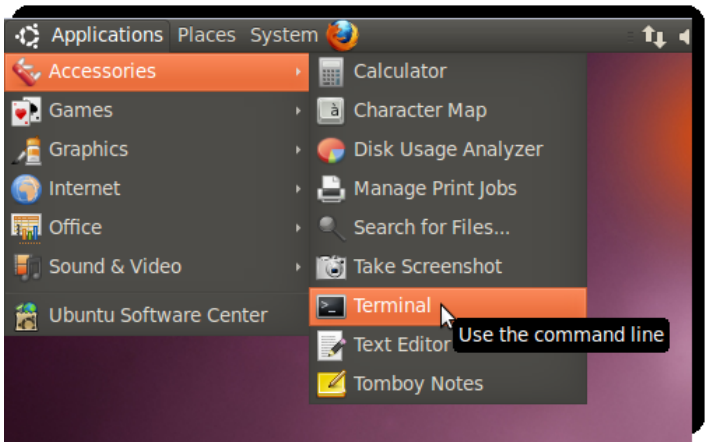



**Figure 4 – An example on Ubuntu Linux of the Terminal being launched**

The terminal screen can be typically accessed from the GUI of most Linux IDEs. It is from here that steps previously described in this document can be executed and the cross-compilation can take place. Delivery of the compiled uCLinux kernel with environment uses a *TFTP* server (*bundled with NetFusion package*) via Ethernet cable, along with CLI command line using the USB-UART.

## 12.2 Source File Structure & NetFusion Package

### 12.2.1 Package Deliverable

From the Nine Ways (www.nineways.co.uk) or Distributor website, you can obtain the *netfusion-uclinux-x.xx.gz* file. The "*x.xx*" is the version of that build.

Use earlier sections in this document to instruct and help you to install the Emcraft uCLinux deployment and then the toolset of the cross-compile inside the target destination. Those are the first two steps of the three required to get the full NetFusion development environment.

Copy the *netfusion-uclinux-x.xx.gz* package file into the *Project/linux-cortexm-1.xx.x/projects* directory that was created from the first Emcraft uCLinux package installation.

Firstly, you have default initial directory folders which need to be deleted:

```
/linuxPC # cd Project/linux-cortexm-1.xx.x/projects
/linuxPC:/ Project/linux-cortexm-1.xx.x/projects # rm * -fr
```

The directory will now be empty ready for the NetFusion package contents. Copy the package gzip file into the projects directory that you downloaded or have kept archived:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects # cp [source dir]/netfusion-uclinux-x.xx.gz *
```

You can use the following command to unpack the package file into the *projects/* dir:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects # gzip -d netfusion-uclinux-x.xx.gz
```

Or alternatively:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects # gunzip netfusion-uclinux-x.xx.gz
```

If the package is a file with an extension of *tar.gz*

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects # tar -xzvf netfusion-uclinux-x.xx.tar.gz
```

You will now have the following contents in the *projects/* directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects # ls
linuxPC:/ Project/linux-cortexm-1.xx.x/projects # netfusion opentftp Rules.make
```

The *netfusion* directory contains all of the make files, configuration files, source files, header files, device drivers, applications, SQLite source and much more for the NetFusion baseboard.

The *opentftp* is the full *TFTP* server used as a stand-alone tool for when the uCLinux kernel binary file is to be uploaded to the NetFusion via the network. This is customized and defaults to the needs of the upload operation. There are scripts in the build system that invokes this service.

The file *Rules.make* is the make build script that is used by the Emcraft uCLinux compile procedure.

### 12.2.2 Host PC Cross-Compile Directory Structure

Change into the *netfusion/* directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects # cd netfusion
```

Look at the contents:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion # ls

local netfusion.busybox netfusion.initramfs netfusion.uImage savetftp starttftp Makefile
netfusion.kernel.M2S netfusion.uboot.s patch source

linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion #
```

***local*** – **sub-directory** – contains files to be copied onto the NetFusion file system

***netfusion.busybox*** – build configuration file – Busybox apps enable/disable configuration file

***netfusion.initramfs*** – build configuration file – defines tree structure of the NetFusion file system

***netfusion.uImage*** – image of compiled NetFusion kernel – pre-compiled image of the kernel file

***savetftp*** – script – copies the compiled kernel to the *TFTP* server root directory

***starttftp*** – startup script – invoked at start of a session to run the *TFTP* server

***Makefile*** – Linking makefile – used for the build process. Add your apps to this.

***netfusion.kernel.M2S*** – build configuration file – uCLinux kernel build options for NetFusion

***netfusion.uboot.s*** – u-boot image file – image of the u-boot used to by NetFusion before uCLinux boot-up

***patch*** – **sub-directory** – contains files to applya NetFusion specific modification to the uCLinux kernel code that is not possible from the *(above)* configuration files

***source*** – **sub-directory** – contains source files, headers file of the NetFusion software apps, device drivers and packages

The two execution script shell files *savetftp* and *starttftp* may need to be modified to allow them to be executable scripts.

Type

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion # chmod +x savetftp
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion # chmod +x starttftp
```

Go into the *local/* directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion # cd local
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/local # ls

default.script group hosts html httpd.conf inittab.ALL passwd rc securetty services shadow
ssh_host_dss_key ssh_host_rsa_key

linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/local #
```

***html*** – **sub-directory** – contains files for the default apache web server

All other files are placed into the */etc* directory on the NetFusion SOM uCLinux environment file system. You can change the contents of the file that get deployed into the */etc* directory of the NetFusion here in this directory and know it will be the default contents of the */etc* files. This is because the root file structure in *ramFS* and defaults back to the start contents for each reboot.

Go into the *html/* directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/local # cd html
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/local/html # ls

aci.cgi aci_driver.html board_checks_main.html datetime_bc.html drivers_main.html
ds1307_driver.html electronic.jpg file.name filesys_bc.html index.html lcd.cgi
lcd_driver.html main.htm mydemo.cgi pcf.cgi pcf_driver.html rtc.cgi sql.cgi tem.cgi vol.cgi
volt_sup_driver.html

linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/local/html #
```

****.jpg*** – image picture file – Used in the demo web page as images

****.html*** – HTML web page file – Secondary web page HTML files referred deeper into the website from the index.html

***Index.html*** – Main root web page – Start root web page when pointing a browser to the NetFusion web server *ip address*.

****.cgi*** – CGI script file – *CGI* files that are invoked from the *HTML* files to action a driver on the NetFusion board which then passes back the information to be displayed. This is a simpler form of using *Python* or *PHP* linking to a database etc.

Go back out and then to the netfusion/*patch*/ directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/local/html # cd ../..
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/ # cd patch
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/patch # ls

applypatch spi.c

linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/patch #
```

The script shell file *applypatch* may need to be modified to allow it to be an executable script.

Type

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/patch # chmod +x applypatch
```

***applypatch*** – shell script file – copies the source files in this directory into the relevant locations that they reside in the uCLinux kernel from Emcraft. Only when a configuration of the OS environment cannot be controlled by the main *[.config]* scripts in the main directory is the patch necessary.

***spi.c*** – C source file – a modified resized image of the *JFFS2* flash on the SPI memory on the SOM. This allows for a much larger uCLinux kernel and NetFusion overall package to be deployed.

Go back out and then to the *source/* directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/patch # cd ..
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/ # cd source
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/patch # ls

aci_ad7998.c aci_ad7998.sh coreUART.c coreUART.sh ds1307rtc.c ds1307rtc.sh factory_test.sh
genGPIO_UIO.ko gen_uio_driver.c include lcd_pcf5074.c lcd_pcf5074.sh Makefile moduleparms
modules.order mtipDMAontroller_dmaCORE.c mtipDMAController_extraHW.c
mtipDMAController_macFIFO.c mtipDMAController_main.c mtipDMAController_master_receive.c
mtipDMAController_slave_receive.c mtipDMAController_switchFIFO.c mtipDMAController_syncgen.c
mtipDMAController_timermod.c mtipDMAController_UIO.c mtipdw_dma_sw.ko mtiptest mtiptest.c
object pcf5074.c pcf5074.sh spi_audio_line_in.c spi_audio_line_in.sh spi_audio_line_out.c
spi_audio_line_out.sh sqlite temp_sensor.c temp_sensor.sh vs_ad7998.c vs_ad7998.sh

linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source #
```

The execution script shell files aci_ad7998.sh, coreUART.sh, ds1307rtc.sh, factory_test.sh, lcd_pcf5074.sh, pcf5074.sh, spi_audio_line_in.sh, spi_audio_line_out.sh, temp_sensor.sh and vs_ad7998.sh may need to be modified to allow them to be executable scripts.

Type

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source # chmod +x *.sh
```

File Descriptions:

| | | |
|---|---|---|
| NetFusion UIO user-space device drivers for driving the PCB baseboard hardware | | |
| aci_ad7998.c | C file | Contact Inputs device driver |
| coreUART.c | C file | RS485 stream device driver |
| ds1307rtc.c | C file | Real-Time clock handler |
| gen_uio_driver.c | C file | Base file that compiles into kernel to allow the rest of these device drivers at the user-space level to attach to a UIO node |
| lcd_pcf5074.c | C file | LCD control device driver |
| mtiptest.c | C file | 'iperf' like, test tool |
| pcf5074.c | C file | General purpose input/output (GPIO) signal driver. Controls LED, relays, GPIO and reads buttons, switches, inputs and option selectors |
| spi_audio_line_in.c | C file | Line IN Stereo audio stream driver |
| spi_audio_line_out.c | C file | Line OUT Stereo audio stream driver |
| temp_sensor.c | C file | PCB mounted Temperature sensor device driver |
| vs_ad7998.c | C file | Voltage monitor driver |
| Included Special AMBA-DMA IP core device driver for linking Ethernet MAC FIFO to SDRAM | | |
| moduleparms | Configuration File | DMA config setup & control file |
| mtipDMAontroller_dmaCORE.c | C file | Core DMA functions in C |
| mtipDMAController_extraHW.c | C file | Additional specific hardware platform setup and configuration C functions such as Microsemi & Altera |
| mtipDMAController_macFIFO.c | C file | MAC interface functions |
| mtipDMAController_main.c | C file | Main start program for the DMA and setup containing IRQ handlers, and call-back functions from the uCLinux kernel IP stack |
| mtipDMAController_master_receive.c | C file | PTP-1588 Clock Sync Functionality |
| mtipDMAController_slave_receive.c | C file | PTP-1588 Clock Sync Functionality |
| mtipDMAController_switchFIFO.c | C file | SWITCH interface functions |
| mtipDMAController_syncgen.c | C file | PTP-1588 Clock Sync Functionality |
| mtipDMAController_timermod.c | C file | PTP-1588 Clock Sync Functionality |
| mtipDMAController_UIO.c | C file | UIO platform for user-space applications to interrogate the DMA IP core with-out using this device driver |
| NetFusion device drivers shell scripts for easier use and examples of executing the drivers of the PCB baseboard hardware | | |
| aci_ad7998.sh | Shell Script | Corresponds to user-space driver |
| coreUART.sh | Shell Script | Corresponds to user-space driver |
| ds1307rtc.sh | Shell Script | Corresponds to user-space driver |
| factory_test.sh | Shell Script | Corresponds to user-space driver |
| lcd_pcf5074.sh | Shell Script | Corresponds to user-space driver |
| pcf5074.sh | Shell Script | Corresponds to user-space driver |
| spi_audio_line_in.sh | Shell Script | Corresponds to user-space driver |
| spi_audio_line_out.sh | Shell Script | Corresponds to user-space driver |
| temp_sensor.sh | Shell Script | Corresponds to user-space driver |
| vs_ad7998.sh | Shell Script | Corresponds to user-space driver |
| Directories with-in the source development tree | | |
| include | Sub-directory | Contains all C file include header files from this directory |
| mtiptest | Sub-directory | Contains the bulk of the 'iperf' styled SOM Ethernet test application |

```
object                                  Sub-directory       Temporary compiler dump directory
sqlite                                  Sub-directory       All SQLite (MySQL-uCLinux) source

AMBA-DMA kernel device driver object generation

genGPIO_UIO.ko                          Kernel Object       Pre-compiled device driver for UIO
                                                            user-space drivers accessing the
                                                            physical hardware
mtipdw_dma_sw.ko                        Kernel Object       DMA Controller kernel device driver
                                                            pre-compiled for use without a re-
                                                            build

uCLinux build script <Makefile> for this local directory nesting

Makefile                                Build script        Main build file script for the outer
                                                            makefile to invoke to compile the
                                                            contents of this directory and the
                                                            sub-directories except the SQLite
                                                            which is separate.
modules.order                           Build script
```

Go back out and then to the *include/* directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source # cd include
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/include # ls

implementation.h lockfile.h mtipDMAController.h mtipDMAController_tsm_defs.h mtipmacFIFO.h
mtipUIO.h mtip1588_syncgen.h mtipextraHW.h mtipswitchFIFO.h

linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/include #
```

***implementation.h*** – C header file – ***#define*** controls for the entire AMBA DMA Controller device driver in the previous directory

***lockfile.h*** – C header file – Used to allow for file-node semaphores across many of the NetFusion device driver applications to protect memory resources from mutli-threaded problems

***mtipDMAController.h*** – C header file – Used as an include for DMA Controller

***mtipDMAController_tsm_defs.h*** – C header file – DMA Controller PTP-1588 include file

***mtipmacFIFO.h.h*** – C header file – MAC FIFO interface C definition for DMA Controller

***mtipUIO.h*** – C header file – UIO memory base access include file for the DMA Controller

***mtip1588_syncgen.h*** – C header file – DMA Controller PTP-1588 include file

***mtipextraHW.h*** – C header file – Specific hardware platform include file for

the AMBA DMA Controller in the previous directory

***mtipswitchFIFO.h*** – C header file – SWITCH FIFO interface C definition for DMA Controller

Go back out and then to the *mtiptest/* directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/include # cd ..
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source # cd mtiptest
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/mtiptest # ls

mtipdefine.h mtipRAWClient.c mtipRAWlo.h mtipTCPClient.c mtipTCPClient.c mtipUDPlo.h
mtipFunctions.c mtipRAWClient.h mtipRAWServer.c mtipTCPClient.h mtipTCPServer.c
mtipUDPCLient.h mtipUDPServer.c mtipFunctions.h mtipRAWlo.c mtipRAWServer.h mtipTCPlo.c
mtipTCPServer.h mtipUDPlo.c mtipUDPServer.h

linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/mtiptest #
```

All of the contents of the *mtiptest* directory are described in detail in the SmartFusion2 User Guide PDF

**Important Note:** *ALL of the <mtipDMAController> (AMBA DMA Controller functionality and files) are described in the* AMBA DMA Controller Linux Device Driver User Guide PDF

### 12.2.1 File List of *mtiptest* Directory

The files in the <*mtiptest*> directory will be named after the test that they will be performing, there are a few exceptions and the list below shows the name and the meaning of all files names.

| Name | Type | Test |
|---|---|---|
| MtipRAWClient.c | Functions Implementation | RAW socket Client test |
| mtipRAWClient.h | Header | RAW socket Client test |
| mtipRAWServer.c | Functions Implementation | RAW socket Server test |
| mtipRAWServer.h | Header | RAW socket Server test |
| mtipRAWlo.c | Functions Implementation | RAW socket Loop Back Test |
| mtipRAWlo.h | Header | RAW socket Loop Back Test |
| mtipTCPClient.c | Functions Implementation | TCP Client test |
| mtipTCPClient.h | Header | TCP Client test |
| mtipTCPServer.c | Functions Implementation | TCP Server test |
| mtipTCPServer.h | Header | TCP Server test |
| MtipTCPlo.c | Functions Implementation | TCP Loop Back Test |
| mtipTCPlo.h | Header | TCP Loop Back Test |
| mtipUDPClient.c | Functions Implementation | UDP Client test |
| mtipUDPClient.h | Header | UDP Client test |
| mtipUDPServer.c | Functions Implementation | UDP Server test |
| mtipUDPServer.h | Header | UDP Server test |
| MtipUDPlo.c | Functions Implementation | UDP Loop Back Test |
| mtipUDPlo.h | Header | UDP Loop Back Test |

**Table 1: App Program File List and Meanings**

*There are a few exceptions from this list:*

| Name | Type | What it does? |
|---|---|---|
| mtipDefine.h | Header | Store the names used in the whole code |
| Mtiptest.c | Main Implementation | Store the Main Function |
| mtipFunctions.h | Header | |
| mtipFunctions.c | Functions Implementation | Functions used by all files |

**Table 2: App Program Exceptions for File List**

#### 12.2.1.1 Understanding the names of the Functions

The main file of the App *mtiptest.c* will read the command line to know if it is going to work as server, client or loop-back, than each test (*UDP, TCP, RAW*) will be called for the selected operational mode. In this case the functions names follow a pattern that will be described in the next table.

Inside the *TCPsvMain*, another function will be called to handle the communication between the endpoints, but this one is not recurrent. Each type of test will have its own function that will call all functions necessary to create sockets, change their options and handle the communication.

| Name | What it does? |
|---|---|
| TCPclMain | TCP Client Main |
| TCPsvMain | TCP Server Main |
| TCPloMain | TCP Loop Back Main |
| UDPclMain | UDP Client Main |
| UDPsvMain | UDP Server Main |
| UDPloMain | UDP Loop Back Main |
| RAWclMain | RAW Client Main |
| RAWsvMain | RAW Server Main |
| RAWloMain | RAW Loop Back Main |
| UDPloReport | UDP Loop Back Report |
| TCPloReport | TCP Loop Back Report |
| RAWloReport | RAW Loop Back Report |
| UDPclReport | UDP Client Report |
| TCPclReport | TCP Client Report |
| RAWclReport | RAW Client Report |

**Table 3: App Program Source content**

### 12.2.1.2 Understanding Variables and Socket names

To name the variables and sockets used in this code, the same standard used to name functions will be used here, the protocol names will be expressed in capital letters (*RAW, UDP, TCP*) and the operational mode will be expressed in two lower-case letters, (*cl, sv, lo*). The next table will show some of the variables used on the code excluding the explanation of which protocol or mode it belongs.

| Name | Functional Description |
|---|---|
| TCPloTotalBytesT0 | Total of bytes received in the first test |
| UDPclPacketRec | Total of packets received |
| SockUDPsv | Name of one of the sockets |
| RAWclTimeTest | Duration of the test |
| speedT0 | Speed of the first test in one of the reports (local variable) |
| RAWclTotalBytes | Total of bytes received in the last test, before applying it to a test number |

**Table 4: App Program Source important variables**

### 12.2.1.3 Changing the Default Settings

To change the *DEFAULT* sever address, MAC address and port numbers the file *mtipDefine.h* has to be edited and the software must be recompiled and exported to the SmartFusion2 again.

Inside this file, change the old information to the new address of port number, as in these two examples.

*To change the Default MAC (below):*

```
61  #ifndef DEFAULTMAC
62  #define DEFAULTMAC  "c0:b1:3c:83:83:83"
63  #endif
```

**Figure 5 – Default MAC Setting**

Change "c0:b1:3c:83:83:83" to your client MAC address.

*To change the DEFAULT server address (below):*

```
53  #ifndef SERVERNAME
54  #define SERVERNAME  "10.1.1.106"
55  #endif
```

**Figure 6 – Default Server Name Setting**

Then change "10.1.1.106" to your Server Address.
Save the file with the same name in the same directory.

### 12.2.2 SQLite Directory

Go to the *sqlite/* directory from the *netfusion/source/* directory:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source # cd sqlite
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/sqlite # ls

backup dlfcn.h Makefile sqlite3.c sqlite3_configure sqlite3ext.h sqlite3.h bits lockfile.h
shell.c sqlite3

linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/sqlite #
```

The execution script shell file sqlite3_configure may need to be modified to allow it to be an executable script.

Type

```
linuxPC:/linux-cortexm-1.xx.x/projects/netfusion/source/sqlite # chmod +x sqlite3_configure
```

***sqlite3_configure*** – configuration initialization shell – run this to discover cross-compile environment for the SQLite build process

***backup*** – **sub-directory** – Old files remaining from before uCLinux Emcraft porting

***bits*** – **sub-directory** – Some Header files

***Makefile*** – build system file – Separate make system to the main Emcraft uCLinux in outer directories. This allows for the separate compilation to produce the *sqlite3* executable

***sqlite3*** – Pre-compiled executable – Cross-compiled SQLite binary

***dlfcn.h*** – C header file – SQLite C-header file

***sqlite3.c*** – C source file – Main 150,000 function containing core of the entire SQLite system. This was ported and merged into a few extremely large files.

***sqlite3ext.h*** – C header file – SQLite C-header file

***sqlite3.h*** – C header file – SQLite C-header file

***lockfile.h*** – C header file – Used to allow for file-node semaphores across many of the NetFusion device driver applications to protect memory resources from mutli-threaded problems

***shell.c*** – C source file – Functions and structures relating to the command line queries and the IO streams of the database contents.

## 12.3 SQLite Usage

The SQLite is a specifically ported uCLinux orientated MySQL query binary application. As discussed, the source code compiles to the binary using *make*. However, once the user is running and using in the target NetFusion uCLinux system, a whole level of support and documentation is needed for those users who are not as familiar with the service.

*Please refer to the following documentation:* SQLite Official Website

Also, type *sqlite3 –help* on the NetFusion prompt for full descriptive options and usage as a command helper.

## 12.4 The AMBA DMA Controller Device Driver

### 12.4.1 What is the AMBA DMA Controller?

The DMA Controller is an IP core that is provided in the Libero starter project that is bundled with the NetFusion product. It is located in the SmartFusion2 FPGA fabric and interfaces via an AHB-Lite memory bus. Below describes the related and associated device driver and the source files that have briefly been covered (*above*).

### 12.4.2 Source Files

The files associated with the AMBA DMA Controller are:

**include <***directory***>**
**object <***directory***>**

Makefile, moduleparms, mtipDMAcontroller_main.c, mtipDMAcontroller_dmaCORE.c, mtipDMAcontroller_extraHW.c, mtipDMAcontroller_macFIFO.c, mtipDMAcontroller_switchFIFO.c, mtipDMAcontroller_UIO.c, mtipDMAController_timermod.c, mtipDMAController_syncgen.c, mtipDMAController_slave_receive.c, mtipDMAController_master_receive.c, mtipsw_dma_s.ko

| | |
|---|---|
| **include** <directory> | - All C header files for the DMA device driver |
| **object** <directory> | - DMA device driver target object file compilation dump |
| **Makefile** | **-** A copy of one of the next two make files below |
| **moduleparms** | **-** DMA device driver settings file upon driver registration |
| **mtipDMAcontroller_main.c**, | - DMA device driver kernel object structured main source |
| **mtipDMAcontroller_dmaCORE.c,** | - Portable *Send(), Receive(), Open(), Close()* functions for DMA main functionality written to be generic. |
| **mtipDMAcontroller_extraHW.c**, | - Portable functions for extra device setup for specific custom hardware for instance bespoke target device initialization via IO Register access (*initialization and shutdown only*). |
| **mtipDMAcontroller_macFIFO.c**, | - Portable functions for the possibility of MAC IP core setup for the attached hardware to the DMA via IO Register access (*initialization and shutdown only*). |

**mtipDMAcontroller_switchFIFO.c**, - Portable functions for the possibility of SWITCH IP core setup for the attached hardware to the DMA via IO Register access (*initialization and shutdown only*).

**mtipDMAController_timermod.c,** - PTP 1588 Clock Synchronization timer interfacing hardware functions

**mtipDMAController_syncgen.c,** - PTP 1588 PTP frame packet generation coding and handling

**mtipDMAController_slave_receive.c,** - PTP 1588 Slave mode of operation functions and code features

**mtipDMAController_master_receive.c,** - PTP 1588 Master mode of operation functions and code features

**mtipDMAcontroller_UIO.c**, - Specific Linux structured and bound source file to allow application software in Linux to send/receive sideband data to the **RX** and **TX** of the DMA device driver and ten through to/from the DMA hardware and *MAC/SWITCH* interconnection.

**mtipsw_dma_s.ko**, - If not cleaned, this file exists as the last *(delivered default)* kernel object of the DMA device driver. This will get deployed into the */lib/* directory on the uCLinux NetFusion file system as part of a dynamic device driver kernel object mechanism.

### 12.4.3 Concept

*The driver for the DMA Controller fulfills the following tasks:*

1. It exposes a regular network device in the Linux stream architecture (ethX, eth[X + 1] or mtipX, mtip[X + 1]) usually with the pneumonic </dev/eth0 /dev/mtip0> or </dev/eth1 /dev/mtip1> that any application layer API program can access. This along with very importantly, the BASH Unix scripting that binds powerful high-level programs to work together. (The configuration file for the driver source allows for the naming of the exposed network device).

   **Note:** *the network interface name "eth" or "mtip" are just ideal default names. They are configurable text wise in the software device driver.*

2. It manages the full control of the data throughput of the DMA master AXI controller invoking memory transfer to/from the DDR physical contiguous SDRAM.
3. It exposes a UIO interface for the API application programs (UioX) to be able to access all of the MorethanIP register mapping of all integrated hardware logic IP cores with-in the silicon device of the ARM processor.
4. Finally, the device driver initializes ALL hardware IP cores additional to the DMA Controller module such as any MAC, PHY, PCS and/or SWITCH cores (however the default deliverable driver usually will only handle DMA registers with basic minimal support for the additional *MAC/SWITCH* hardware). The aim is not to provide full operational control and management but purely for basic information gathering such as Link *UP* and *DOWN* and flow control error checking via IRQ Interrupt Service Routines (ISRs).

The driver is implemented in a kernel module named **MTIP-DMA-SW.**

### 12.4.4 Driver Operation & Functional Description

The Linux kernel device driver named **[mtipsw-dma-sw]** is an installable independent module written in the C-programming language. It is considered separate in meaning because it is not part of the compiled kernel or available in the mainline released kernel from **<*www.linux.org*>.** In fact this device driver that is bundled with the DMA Controller comes packaged as C source code with [*make*] file scripts for GCC Linux. It allows the Ethernet frame to/from the DMA hardware connected to a *MAC/SWITCH* to pass to a network interface in uC-Linux. From here, it then allows as standard, application programs to communicate with a network via standard sockets using IP and other allowed upper layer protocols. The application programs run as threads in uCLinux utilizing all the powerful features of such a well established OS.

The relevance of distributing as source is to allow as standard with in a Linux environment, for the compiler and linker to detect and adjust for the host architecture and surrounding kernel structure. As there is no specific kernel that is needed (although stay more recent than ***version 2.6.14***) the compilation and build of the device driver source to a kernel object file will fit to its surrounds. In the event that a significant step in kernel produces a compilation error, then *you will have to contact MorethanIP GmbH or Nine Ways R&D Ltd for support*. (www.nineways.co.uk).

### 12.4.5 High and Low Priority Channels

As already discussed the DMA Controller concept is built on a dual RX and dual TX channel mechanism. Basically, from the FIFO interface on a *MAC/SWITCH* all the way to the application software programs in the processor sub-system, a dual channel pathway in each direction exists offering overtaking of packet frames.

This sounds unnecessary as the frame usually will stay in order along the route of a fiber router hop (unless the higher layer router functionality obeys IP fragmentation rules). However, the section of the pathway in between a given MAC and the software applications handling the data/information contained in the Ethernet packets is often bottlenecked and the slowest part of the whole process.

In order to allow for this section of restriction to be less hindering to chosen packets, two pathways for the TX system and two pathways for the RX system allow for priority to take place and thus, lower priority Ethernet frames are held up in their lanes whilst the higher priority are transacted through first.

The Linux software device driver shall provide for the two lanes per direction. Thus CH0, CH1 for RX-DMA. Also CH0, CH1 for TX-DMA.

No other assumptions or priority takes place with the device driver as the hardware reads the TX-DMA first and the IP stack upper layers in the OS can decide themselves the priority of the reading of the RX-DMA from the descriptor information.

The device driver just has to provide the parallel channels in both directions.

### 12.4.6 Linux/uC-Linux Network Interface Exposure

The device driver will cause the Linux/uC-Linux OS to expose two separate network interfaces to the application layers for the DMA Controller connected to a *MAC/SWITCH*.

**Note:** *Whilst the name of the exposed network interfaces is configurable (described below) it will be illustrated and described as /dev/eth0 and /dev/eth1 here in this sub-section.*

The CH0 for RX and TX together will be piped through to exposed interface */dev/eth0*. This is easy to remember as *eth0* is CH0 high priority.

Also, CH1 for RX and TX together will be piped through to exposed interface */dev/eth1*.

At this point, it is widely assumed that the software application programs that *bind()* to selected network interfaces will ultimately control how the TX-DMA effectively utilizes this mechanism. The RX-DMA will automatically tend to (according to the hardware specification in the *MAC/SWITCH* IP cores) prioritize and order the packets arriving. This means the upper layer software is not aware of the prioritization.

However, it is for the system architect and designer to ensure that the different software using different Ethernet traffic types to ensure that the application programs are binding to the relevant exposed network interfaces.

Usually, bridge protocol packets are assigned higher priority and so they could be using */dev/eth0* and the RSTP demon can assign bridge <*br0*> to use this network interface. This allows for those BPDU packets to be processed much higher priority as the network topology has to be prioritized to a much more important extent that the normal Ethernet frame data traffic.

This whole mechanism is to enforce packet type prioritization from end point software applications, through DMA, MAC interfaces, switches and routing. Not all of this can often be achieved but for this product, the DMA Controller is fully ready for the functionality requirement of channel priority.

### 12.4.7 DMA Controller Device Driver Parameter List

The parameters for the uCLinux AMBA DMA Controller device driver are kept in **<moduleparms>** file in the */lib/* directory of the delivered package. This is a text file but is not a C header file or any other programming language format. In fact, it is a standard scripting format that is piped into **<xargs>** and **<insmod>** Linux commands upon device driver registration and instantiation.

## 12.5 Building the NetFusion Customized uCLinux Kernel

The entire NetFusion uCLinux environment can be built from clean in two simple steps. For a normal incremental make which just updates any changes, then it can be generally built in a single step. However, if the changes are being made to SQLite then it is once again two steps as opposed to a single make process. If you have just booted your PC, or have closed down a terminal session and then re-started, you will have to go through an initialization step first. You must activate your uCLinux Cortex-M development session by navigating to the top of your uCLinux Cortex-M installation and run:

```
[psl@pvr linux-cortexm-1.12.0]$ source ACTIVATE.sh
```

Then change into the *netfusion/* directory

```
linuxPC:/ Project/linux-cortexm-1.xx.x/ # cd projects/netfusion
```

Also, if you had to run the *source ACTIVATE.sh* (*above*), then you must also run the following:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/ projects/netfusion # ./startftftp
```

To clean the entire project type:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion # make clean
```

Then to clean the SQLite separate build system type:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion # cd source/sqlite
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/sqlite # make clean
```

Then make the SQLite by typing:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/sqlite # make
```

Then make the full uCLinux build with NetFusion (including the SQLite3 binary) by typing:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/sqlite # cd ..
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source # cd ..
liuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion # make
```

Finally, prepare the newly generated uCLinux kernel package binary for TFTP upload:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/ # ./savetftp
```

### 12.5.1 Building the SQLite Incrementally

Then change into the *netfusion/source/sqlite* directory and build the binary with *make*

```
linuxPC:/ Project/linux-cortexm-1.xx.x/ # cd projects/netfusion/source/sqlite
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/source/sqlite # make
```

### 12.5.2 Building the main uCLinux with NetFusion Incrementally

Then change into the *netfusion/* directory and build the binary with *make*

```
linuxPC:/ Project/linux-cortexm-1.xx.x/ # cd projects/netfusion
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion # make
```

Finally, prepare the newly generated uCLinux kernel package binary for TFTP upload:

```
linuxPC:/ Project/linux-cortexm-1.xx.x/projects/netfusion/ # ./savetftp
```

### 12.5.3 Uploading the kernel to the NetFusion

To load the image directly, use the *netboot* U-boot macro:

```
M2S-SOM> setenv image netfusion.uImage
M2S-SOM> run netboot
...
TFTP from server 172.17.0.1; our IP address is 172.17.5.100
Filename 'networking.uImage'.
...
Loading: #################################################################
         #################################################################
         #############
done
Bytes transferred = 2084704 (1fcf60 hex)
...
   Image Name:   Linux-2.6.33-arm1
   Image Type:   ARM Linux Kernel Image (uncompressed)
...
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

Linux version 2.6.33-arm1 (vlad@ocean.emcraft.com) (gcc version 4.4.1 (Sourcery G++ Lite
2010q1-189) ) #1 Fri Dec 06 15:43:44 MSK 2013
...
```

To load the image into the flash, use the u-boot update macro:

```
M2S-SOM> setenv image netfusion.uImage
M2S-SOM> run update
...
TFTP from server 172.17.0.1; our IP address is 172.17.5.100
Filename 'networking.uImage'.
...
Loading: ##################################################################
         ##################################################################
         #############
done
Bytes transferred = 2084704 (1fcf60 hex)
16384 KiB S25FL128S_64K at 0:0 is now current device
Saving Environment to SPI Flash...
Erasing SPI flash...Writing to SPI flash...done
M2S-SOM>
```

Reset the NetFusion baseboard and verify that the newly programmed image boots in the autoboot mode:

```
M2S-SOM> reset
resetting ...

U-boot 2010.03-linux-cortexm-1.12.0 (Dec 06 2013 - 17:19:37)
...
Starting kernel ...
...
init started: BusyBox v1.17.0 (Dec 06 2013 - 17:21:30)
~ #
```

## 12.6 Board uCLinux Target Directory Structure

Separate to the file structure on your host cross-compilation PC *(above previous section)*, it is important to know the default file system tree of the NetFusion SOM uCLinux running on the embedded hardware. This is the file system that your embedded application software will see and use on the NetFusion.

The entire initial file system is started from a default state at boot-up that is unpacked from the *initramfs* system in the uCLinux kernel. This means that each time the NetFusion re-starts or powers-up, the file changes in the main root tree are erased and defaulted back to a known state defined by the cross-compilation contents of the *rc/* directory in the *netfusion/* project folder. The exception to this is the /flash directory on the NetFusion file system which is separately mounted to the JFFS2 flash partition of the SPI flash hardware on the SOM. In this *flash/* directory all files and changes are kept until they are deliberately erased or modified. This is the location that can be used for instance with, SQLite databases, settings files, debug logs, stored data of other kinds.

We can have a look at the root *filesystem /*:

```
~ # ls

bin etc httpd mtiptest sql var dev factory init proc sys drivers flash lib root usr

~ #
```

| | | |
|---|---|---|
| **bin** | **- directory-** | contains Emcraft uCLinux binary executables such as *busybox* |
| **etc** | **- directory-** | has *inittab*, *hosts, passwd* and configuration Linux files for the system |
| **httpd** | **- directory-** | specific NetFusion web server files and *scripts*, *cgi* and *html* |
| **mtiptest** | **- directory-** | the Nine Ways/MorethanIP test *app* for any Ethernet cores which is like '*iperf*' and allows for throughput measurement testing. *iperf* is quite un-portable to this uCLinux and so this custom application has been designed and written for this specific environment. |
| **sql** | **- directory-** | SQLite3 database query executable |
| **var** | **- directory-** | general normal Linux system directory |
| **dev** | **- directory-** | NetFusion devices listed as file nodes |
| **factory** | **- directory-** | Factory manufacturer's test scripts for self-diagnostics |
| **init** | **- directory-** | contains the *init* binary that is first booted by the uCLinux kernel |
| **proc** | **- directory-** | uCLinux kernel read only file nodes representing the kernel's status |
| **sys** | **- directory-** | structure of classes and UIO devices registered in the system |
| **drivers** | **- directory-** | NetFusion UIO driver binaries and associated shell scripts for the PCB baseboard hardware. The shell scripts give examples and show usage more clearly. |
| **flash** | **- directory-** | SPI flash temporary working directory (*do not use*) |
| **lib** | **- directory-** | default installed uCLinux user device drivers. This is defaulted to the AMBA DMA controller and also the UIO platform for the NetFusion user-space drivers. |
| **root** | **- directory-** | home directory of the default root account |
| **usr** | **- directory-** | uCLinux *usr* directory that has symbolic reference links to other nested deeper areas of the filesystem, contains secondary binaries commands and also sharing applications |

Change to the */bin/* directory:

```
~ # cd bin
/bin # ls

aci bash busybox cat chmod chown ..... uname vi volt wc wget xargs

/bin #
```

Change to the */etc/* directory:

```
/bin # cd /etc
/etc # ls

dropbear hosts initab rc services group httpd.conf passwd securetty shadow

/etc #
```

Change to the */httpd/html/* directory:

```
/etc # cd /httpd/html
/httpd/html # ls

Untitled.png aci.cgi aci_driver.html board_checks_mani.html cgi-bbin contact_details.html
datetime_bc.html drivers_main.html . . . . . styles.css tem.cgi temp_sensor_driver.html
tx.txt vol.cgi volt_sup_driver.html

/httpd/html #
```

Change to the */mtiptest/* directory:

```
/ httpd/html # cd /mtiptest
/ mtiptest # ls

test

/ mtiptest #
```

Change to the */sql/* directory:

```
/ mtiptest # cd /sql
/ sql # ls

sqlite3

/ sql #
```

Change to the */var/* directory:

```
/ sql # cd /var
/ var # ls

config lock log run tmp

/ var #
```

Change to the */dev/* directory:

```
/ var # cd /dev
/ dev # ls

console kmem mem mtd0 mtd0ro mtd1 mtd1ro mtd2 mtd2ro mtd3 mtd3ro mtdblock0 mtdblock1
mtdblock2 mtdblock3 mtdblock4 mtdblock5 mtdblock6 mtdblock7 null ptmx pts random sample tty
ttyS0 ttyS1 ttyS2 uio0 uio1 urandom zero

/ dev #
```

Change to the */factory/* directory:

```
/ dev # cd /factory
/ factory # ls

factory_test.sh

/ factory #
```

Change to the */init/* directory:

```
/ factory # cd /init
/ init # ls

init

/ init #
```

Change to the */proc/* directory:

```
/ init # cd /proc
/ proc # ls

1 10 11 12 13 14 2 42 5 6 7 8 83 9 buddyinfo bus cmdline config.gz cpuinfo crypto devices
diskstats driver execdomains filesystems fs interrupts iomem ioports irq key-users keys kmsg
loadavg locks maps meminfo misc modules mounts mtd net pagetypeinfo partitions self slabinfo
softirqs stat sys timer_list timer_stats tty uptime version vmstat zoneinfo

/ proc #
```

Change to the */sys/* directory:

```
/ proc # cd /sys
/ sys # ls

block class devices fs module bus dev firmware kernel

/ sys #
```

Change to the */drivers/* directory:

```
/ sys # cd /drivers
/ drivers # ls

aci_ad7998 aci_ad7998.sh coreUART coreUART.sh ds1307rtc ds1307rtc.sh lcd_pcf5074
lcd_pcf5074.sh milli pcf5074 pcf5074.sh spi_audio_line_in spi_audio_line_in.sh
spi_audio_line_out spi_audio_line_out.sh temp_sensor tem_sensor.sh vs_ad7998 vs_ad7998.sh

/ drivers #
```

Change to the */lib/* directory:

```
/ drivers # cd /lib
/ lib # ls

genGPIO_UIO.ko mtipsw_dma_sw.ko mtipsw_parms

/ lib #
```

Change to the */usr/* directory:

```
/ lib # cd /usr
/ usr # ls

bin share

/ usr #
```

### 12.6.1 NetFusion Web Server

The NetFusion default uCLinux bundle (*as shown above*) has a working and ready-to-run *apache2* web-server. This is compiled and guised in the form of *httpd* from *busybox*. The */httpd/html* directory contains the web page echo-system where you can change and re-design to you own desires and architectural requirements.

As a start point in the bundled NetFusion uCLinux package, the web page system shows a demonstration of the PCB hardware and the SQLite capabilities. Also, the NetFusion user-space UIO drivers are all used to facilitate the web content which can aid and help the user see how the *cgi* mechanism by default, can allow hardware to interact with the web front end. Moreover, you can then grasp the value of the SQLite database system be able to use the hardware and the web-page to create a truly useful product which all takes advantage of the */flash* mounted SPI hardware using the *JFFS2* partition to allow for the persistent storage.

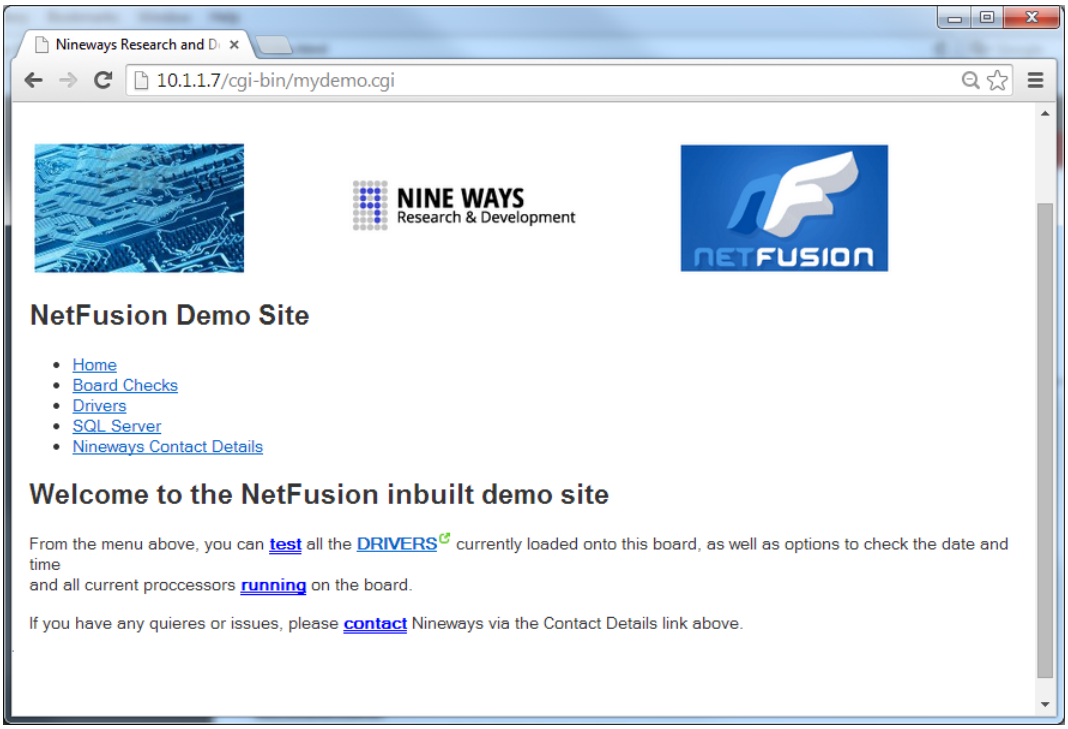

Nineways Research and D
10.1.1.7/cgi-bin/mydemo.cgi
NINE WAYS
Research & Development
NetFusion Demo Site
Home
Board Checks
Drivers
SQL Server
Nineways Contact Details
Welcome to the NetFusion inbuilt demo site
From the menu above, you can test all the DRIVERS currently loaded onto this board, as well as options to check the date and time
and all current proccessors running on the board.
If you have any quieres or issues, please contact Nineways via the Contact Details link above.

**Figure 7 - Main opening screen for the NetFusion Demo Website**

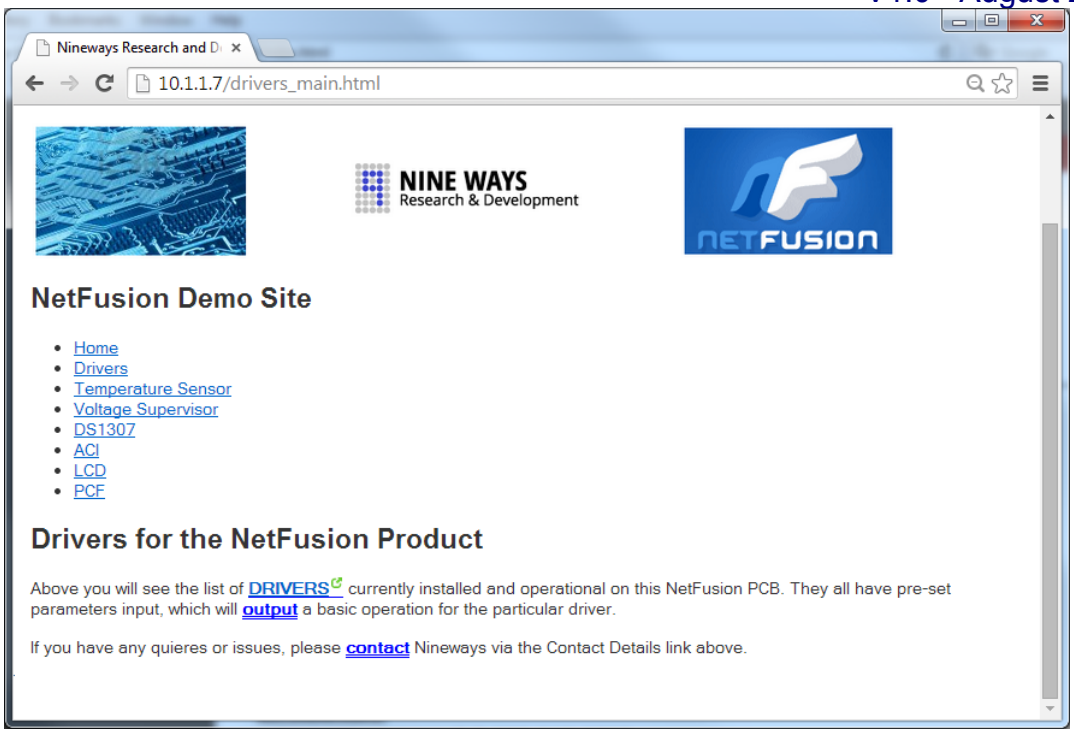

Nineways Research and D
10.1.1.7/drivers_main.html
NINE WAYS
Research & Development
NetFusion Demo Site
Home
Drivers
Temperature Sensor
Voltage Supervisor
DS1307
ACI
LCD
PCF
Drivers for the NetFusion Product
Above you will see the list of DRIVERS currently installed and operational on this NetFusion PCB. They all have pre-set parameters input, which will output a basic operation for the particular driver.
If you have any quieres or issues, please contact Nineways via the Contact Details link above.

**Figure 8 – Drivers/Hardware Page for the NetFusion Demo Website**

# 13 After First Compilation, Getting the System Customized for the User

This is a very important step and thus, is contained in a main-section heading to make this information easily obtainable.

Once you have compiled and downloaded the uCLinux NetFusion image to the baseboard, you can reboot and play. There will be a demo website, all the binary command at your disposal, SQLite, lots of discovering the setup including in the */etc/* directory.

When you have familiarized all of the system and cross-referenced most of this document to the practical experimentation, using the NetFusion product, then you may be comfortable to start planning your customization. This is where you take your copy of the installed bundled into your own project creating your own unique design product.

In order to control the boot-up driver and application launching that is defaulted to a mandatory scheme for you before you think about modifying, the edit the */etc/rc* configuration file

*The contents are listed below:*

```
#!/bin/sh
mount       -t proc proc /proc                  # mount the /proc directory into the filesystem
mount       -t sysfs sysfs /sys                 # mount the /sysfs directory into the filesystem
mount       -t devpts none /dev/pts             # mount the /pts directory into the filesystem
mount       -t jffs2 /dev/mtdblock2 /flash      # mount the /flash directory into the filesystem - non-volatile area
echo         1 >> /proc/sys/kernel/panic        # set the Linux kernel panic time to 1 second. A panic will reboot

ifconfig     lo 127.0.0.1 up                    # bring up the loopback 127.0.0.1 netif
cat          /lib/mtipsw_parms | sed -e 's/#.*$//' -e '/^$/d' | xargs insmod /lib/mtipsw_dma_sw.ko # launch DMA driver

ifconfig    mtip0 down hw ether $ethaddr        # default down mtip0 high-priority netif and set mac address
ifconfig    mtip1 down hw ether $ethaddr        # default down mtip1 lower-priority netif with same mac address
ifconfig     mtip0 10.1.1.190 up                # bring up the new DMA controller CH0 network interface [mtip0]
ifconfig     mtip1 10.1.1.191 up                # bring up the new DMA controller CH1 network interface [mtip1]
insmod      /lib/genGPIO_UIO.ko                 # launch general GPIO-UIO device driver
ifconfig                                        # show on debug output the current network interfaces present

#/usr/app

busybox httpd -h /httpd/html/
/drivers/temp_sensor -file /httpd/html/ts.txt &
```

If you do not want to use or invoke the AMBA DMA Controller device driver because you do have removed it from the SmartFusion2 FGPA fabric, then you can comment out the following lines with the ***"#"*** symbol:

```
# cat        /lib/mtipsw_parms | sed -e 's/#.*$//' -e '/^$/d' | xargs insmod /lib/mtipsw_dma_sw.ko # launch DMA driver
# ifconfig   mtip0 down hw ether $ethaddr        # default down mtip0 high-priority netif and set mac address
# ifconfig   mtip1 down hw ether $ethaddr        # default down mtip1 lower-priority netif with same mac address
# ifconfig    mtip0 10.1.1.190 up                # bring up the new DMA controller CH0 network interface [mtip0]
 # Ifconfig   mtip1 10.1.1.191 up                # bring up the new DMA controller CH1 network interface [mtip1]
```

If you do not want to use or invoke the UIO GPIO core device driver which is used for ALL of the NetFusion user-space drivers (*although this is not a sensible idea*), then you can comment out the following line with the ***"#"*** symbol:

```
# insmod        /lib/genGPIO_UIO.ko                    # launch general GPIO-UIO device driver
```

If you wish to invoke and use the "iperf" styled test app for checking the throughput of the Ethernet ports device drivers (and IP cores) then comment back in the following line of code:

```
/usr/app
```

Of course this can just be called in the command line shell at any time but this will start the process at boot-up time automatically.

If you do not want to use the demo site web-server and have no HTTP access at all to the NetFusion, then you can comment out the following line with the ***"#"*** symbol:

```
# busybox httpd -h /httpd/html/
```

If you do not want to have the NetFusion polling and reading the on-board temperature sensor, then you can comment out the following line with the ***"#"*** symbol:

```
# /drivers/temp_sensor -file /httpd/html/ts.txt &
```

For any other customization, it requires new code, builds, applications and device drivers.

**Happy Coding!**

# 14 NetFusion Board Drivers

## 14.1 uCLinux Application UIO

Netfusion currently has *nine* user-space hardware drivers. They are compiled as applications all separate to each other and are structured and designed such that they interface to the NetFusion PCB hardware via resource protected memory accessible FPGA IP cores (such as the *GPIO* and *coreUART* etc).

At the top of the driver, they communicate either using the input and output streams (or error stream as well) or if necessary, just with command line parameter control and error code return values. However, all drivers have the common hierarchy of access hardware via UIO memory addressing and interpret and format the data through the shell streams or command line.

The UIO device driver */lib/genGPIO_UIO.ko* which is invoked as default by the start-up process (*unless the user removes it by choice*) operates at the uCLinux kernel level. It provides the user-space drivers with a UIO connection down into accessing the physical memory. This in turn, enables the user drivers to controls the I/O lines, buses and IP core hardware.

The driver establishes the <UioX> interface between the hardware and the user-space. During normal operation, i.e. when the user-space application accesses the hardware, no additional software layers are involved. Memory accesses performed by the software are directly interchanged with the hardware.

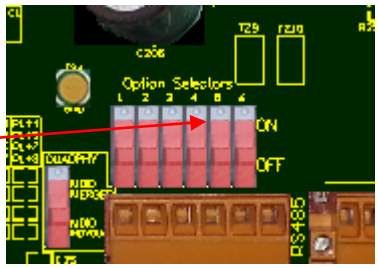
## 14.2 NetFusion User-Space Driver List

| Name of Driver | Source File | Description |
|---|---|---|
| Analogue Contacts (ACI) | *aci_ad7998.c* | Read analogue voltages from connector input |
| RS485 UART | *coreUART.c* | Data in\|out through RS485 bus interface |
| Real-Time Clock (RTC) | *ds1307rtc.c* | Separate PCB baseboard battery-backed clock |
| LCD Controller | *lcd_pcf5074.c* | LCD optional display controller of text and format |
| General I/O (GPIO) | *pcf5074.c* | Individual spare pins and the LEDs, Switches & Jumpers |
| Line-OUT Stereo Audio | *spi_audio_line_out.c* | Stream digital audio to analogue Line-OUT audio |
| Line-IN Stereo Audio | *spi_audio_line_in.c* | Line-IN audio analogue to digital output stream |
| PCB Temperature Sensor | *temp_sensor.c* | An on-board temperature sensor on the baseboard |
| Voltage PCB Supervisor | *vs_ad7998.c* | Monitor and check all voltages and rails on the PCB |

**Table 5 : Full NetFusion User Space Hardware Drivers**

## 14.3 Option Selector Switches

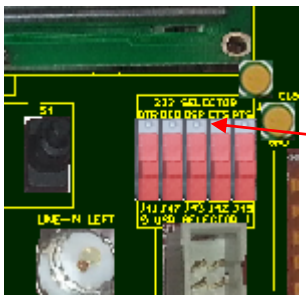
The UP position labeled "ON" produces a "1" on the software read data. Oppositely, the DOWN position labeled "OFF" produces a "0" on the software read data.




Driver:

| | | |
|---|---|---|
| General I/O (GPIO) | *pcf5074.c* | Individual spare pins and the LEDs, Switches & Jumpers |

## 14.4 RS232 / GPIO Extended Signals



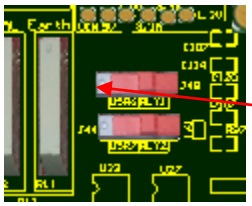
For each of the additional extra RS232 signals such as DTR, DSR etc, they can be switched instead to the GPIO connector if the RS232 functionality is not required.

UP position = RS232 signal.
DOWN position = GPIO USR signal

Driver:

| | | |
|---|---|---|
| General I/O (GPIO) | *pcf5074.c* | Individual spare pins and the LEDs, Switches & Jumpers |

## 14.5 Relays and the Selector Switches



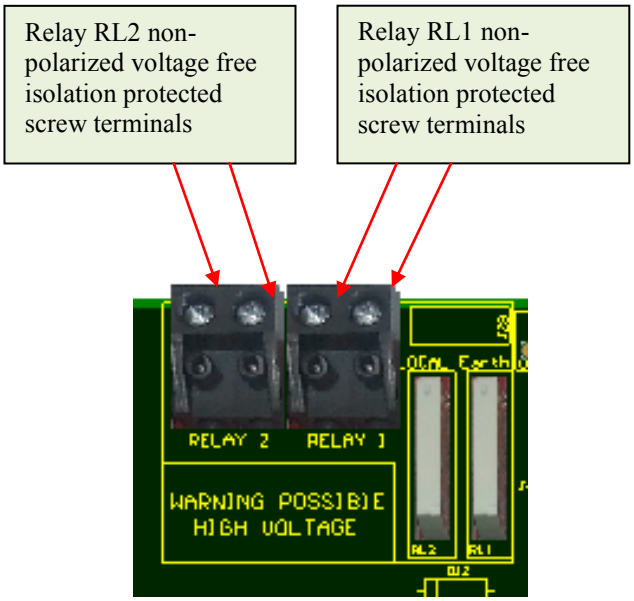
For each of the Relays RL1 and RL2, they can be switched instead to the GPIO connector if the relay output functionality is not required.

LEFT position = GPIO USR signal
RIGHT position = RELAY operation

Driver:

| | | |
|---|---|---|
| General I/O (GPIO) | *pcf5074.c* | Individual spare pins and the LEDs, Switches & Jumpers |

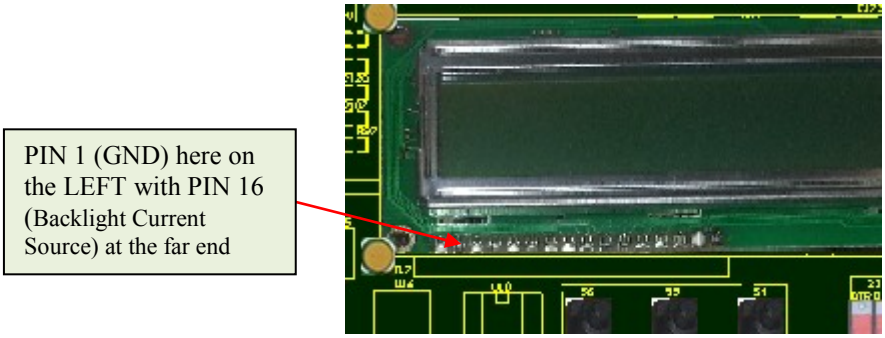
## 14.6 Relay Output Connector


Relay RL2 non-polarized voltage free isolation protected screw terminals
Relay RL1 non-polarized voltage free isolation protected screw terminals
RELAY 2
RELAY 1
WARNING POSSIBLE HIGH VOLTAGE

Driver:

| | | |
|---|---|---|
| General I/O (GPIO) | *pcf5074.c* | Individual spare pins and the LEDs, Switches & Jumpers |

## 14.7 LCD Connector




Driver:

| | | |
|---|---|---|
| LCD Controller | *lcd_pcf5074.c* | LCD optional display controller of text and format |

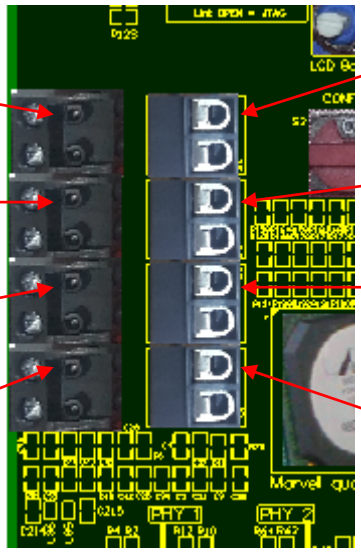
## 14.8 Analogue Contact Input Connector

ACI 1 signal pin 1 pointed to by arrow. GND is the other screw pin (pin 2)

ACI 2 signal pin 1 pointed to by arrow. GND is the other screw pin (pin 2)

ACI 3 signal pin 1 pointed to by arrow. GND is the other screw pin (pin 2)

ACI 4 signal pin 1 pointed to by arrow. GND is the other screw pin (pin 2)



ACI 5 signal pin 1 pointed to by arrow. GND is the other screw pin (pin 2)

ACI 6 signal pin 1 pointed to by arrow. GND is the other screw pin (pin 2)

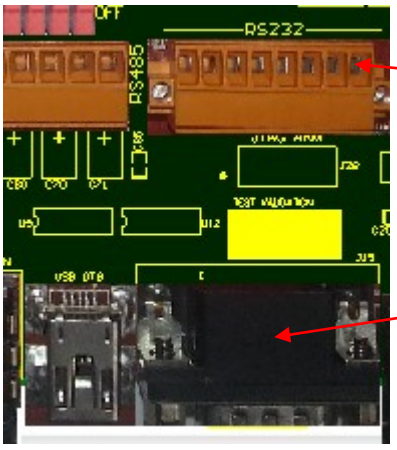
ACI 7 signal pin 1 pointed to by arrow. GND is the other screw pin (pin 2)

ACI 8 signal pin 1 pointed to by arrow. GND is the other screw pin (pin 2)

Driver:

| | | |
|---|---|---|
| Analogue Contacts (ACI) | *aci_ad7998.c* | Read analogue voltages from connector input |

## 14.9 RS232 UART Serial Connectors




8-way screw terminal header RS232 with pin 1 on the right (indicated by the arrow).

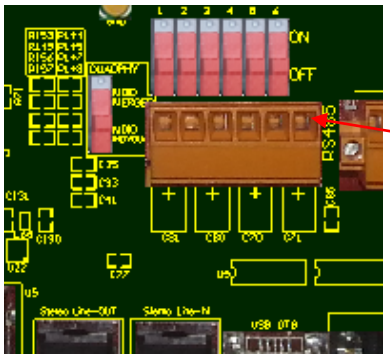
This is electrically in parallel and the same as the DB9 connector below.

This is electrically in parallel and the same as the above connector. It exposes a Male 9-way connector for normal serial cable attachment.

Driver:

Uses standard */dev/ttyS1* node stream in the uCLinux system. This has no specifically written NetFusion driver as it is statutory

## 14.10 RS485 UART Connector

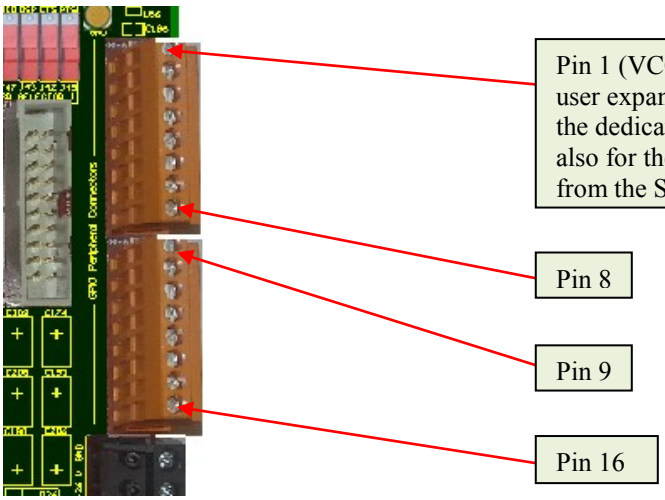


The RS485 screw terminals provide a full-duplex industrial bus system. Pin 1 is indicated by the arrow with pin 6 at the other end.

Driver:

| | | |
|---|---|---|
| RS485 UART | *coreUART.c* | Data in\|out through RS485 bus interface |

## 14.11 GPIO Connector



Pin 1 (VCC3). These pins provide a user expansion connector for some of the dedicated signals on this PCB and also for the rest of the spare GPIO from the SOM headers.

Pin 8

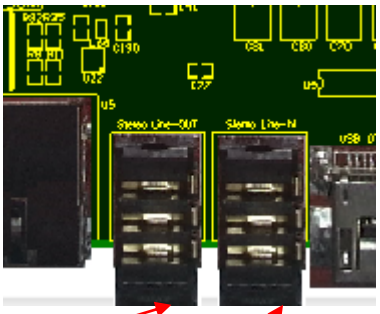
Pin 9

Pin 16

Driver:

| | | |
|---|---|---|
| General I/O (GPIO) | *pcf5074.c* | Individual spare pins and the LEDs, Switches & Jumpers |

## 14.12 Stereo Audio Phono-Jacks



Line Out 4-pin 3.5mm Stereo Jack. The contacts on the PCB are labeled in the below tables

Line In 4-pin 3.5mm Stereo Jack. The contacts on the PCB are labeled in the below tables

Driver(s):

| | | |
|---|---|---|
| Line-OUT Stereo Audio | *spi_audio_line_out.c* | Stream digital audio to analogue Line-OUT audio |
| Line-IN Stereo Audio | *spi_audio_line_in.c* | Line-IN audio analogue to digital output stream |

## 14.13 NetFusion Hardware Drivers Usage

### 14.13.1 Analogue Contacts (ACI)

| | | |
|---|---|---|
| Analogue Contacts (ACI) | *aci_ad7998.c* | Read analogue voltages from connector input |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/aci_ad7998.sh*

*Usage & Syntax*

aci_ad7998 parameter syntax (Must be 10 parameters with '-' switches):
-dev <addr> -sdabit <bit> -clkbit <bit> -gpioaddr <addr> -convbit <bit>

| | |
|---|---|
| *-gpiooffs* | is the user memory offset within the UIO mechanism phys memory of the i2c GPIO signal |
| *-dev* | is the i2C address of the ad7998 IC device on the i2c bus. Check circuit schematics |
| *-sdabit* | is the bit position within the i2caddr location of the sda signal. Zero based between 0-31 |
| *-clkbit* | is the bit position within the i2caddr location of the clk signal. Zero based between 0-31 |
| *-convbit* | is the bit position within the convaddr location of the conv signal. Zero based between 0-31 |

Example: aci_ad7998 -dev 33 -gpiooffs $GPIO_ADDR (Global definition of the GPIO address)
-sdabit 27 -clkbit 26 -convbit 19

*Support: www.nineways.co.uk; lglenn@nineways.co.uk*

*Description*

The ACI Analogue Contact Input driver is responsible for reading the voltage levels of each of the 8 x connector inputs all reference the GND. The results are presented as text values between 0 – 255. Zero is 0V and 255 are 5V. This driver and the hardware are incredibly useful with regards to reading the status of other equipment's relay outputs, door contacts, sensors etc. With analogue inherently built into the core capability, you can design into your product monitored contact functionality to check for temper and sabotage features using series and parallel resistors. Also, this can be used to poll the voltage levels several hundred times per second so this has a lot of other useful added useful purposes.

*Typical return string:* **4, 56, 255, 255, 0, 0, 1, 254\n**

### 14.13.2 RS485 UART

| | | |
|---|---|---|
| RS485 UART | *coreUART.c* | Data in\|out through RS485 bus interface |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/coreUART.sh*

*Usage & Syntax*

coreUART parameter syntax (Must be 10 parameters with '-' switches):
-baudrate <baudrate> -databits <7|8> -baseaddr <baseaddr> -clk <clk> -parity <off|odd|even>

| | |
|---|---|
| *-baudrate* | is the number of bytes per second transferred through the port |
| *-databits* | is used to define the number of valid data bits per byte |
| *-baseaddr* | is used to define the base address of the coreUART in memory |
| *-clk* | is the system clock frequency in hertz |
| *-parity* | is enabled/disabled with the input odd or even or off |

**Example**: coreUART -baudrate 115200 -databits 8 -baseaddr 1929379840 -clk 166000000 -parity off

To exit the driver, input '+++' on the command line

*Support: www.nineways.co.uk; lglenn@nineways.co.uk*

*Description*

The RS485 protected and isolated long distance differential full-duplex 4-wire bus is controlled by a soft-core UART. This is accessed via memory address from this user-space driver and the data to/from the UART core is passed/received using not the command-line, but in fact the read and write streams. The command-line parameters setup the configuration and settings of the session, and then the driver remains in execution whilst shell scripts or other programs can send/receive data through the read and write I/O streams to the driver application.

By sending the escape characters '+++' to the write stream, the driver is forced to exit. Whilst the driver is running, up to exit, the RS485 bus is active and RX data is presented to a controlling program reading the stream from this driver. Oppositely, data sent to the write stream is presented on the hardware TX bus differential pair to form a proper full-duplex system.

*Typical shell script instantiation:* **transmit_queue > coreUART -baudrate 115200 -databits 8 -baseaddr 1929379840 -clk 166000000 -parity off > receive_queue &**

### 14.13.3 Real-Time Clock (RTC)

| | | |
|---|---|---|
| Real-Time Clock (RTC) | *ds1307rtc.c* | Separate PCB baseboard battery-backed clock |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/ds1307rtc.sh*

*Usage & Syntax*

ds1307rtc parameter syntax (Must be more than 15 parameters with '-' switches):
-gpiooffs <addr> -sdabit <bit> -clkbit <bit> -deviceid <bit> -time <hh:mm:ss> -date <dd-mm-yy> -dow <string> -r|-w

| | |
|---|---|
| *-gpiooffs* | is the user memory offset within the UIO mechanism phys mem of the i2c RTC signal |
| *-deviceid* | sets the device ID of the I2C RTC device |
| *-sdabit* | is the bit position within the i2caddr location of the sda signal. Zero based between 0-31 |
| *-clkbit* | is the bit position within the i2caddr location of the clk signal. Zero based between 0-31 |
| *-time* | set the time that will be written for the first time in the RTC. Format hh:mm:ss |
| *-date* | set the date that will be written for the first time in the RTC. Format dd-mm-yy |
| *-dow* | set the day of week that will be written for the first time in the RTC. All letters low case |
| *-r* | read the RTC memory. Do not need a parameter |
| *-w* | write the RTC memory. Do not need a parameter\ |

**Example** (read): ds1307rtc -sdabit 2 -clkbit 3 -gpiooffs 1879048192 -deviceid 104 -r

**Example** (write): ds1307rtc -sdabit 2 -clkbit 3 -gpiooffs 1879048192 -dow friday -time 11:11:11 -date 11-11-11 -deviceid 104 -w

*Support: www.nineways.co.uk; pbates@nineways.co.uk*

*Description*

The NetFusion PCB baseboard has an independent battery-backed 3 year clock that can be accessed by the application software. The SmartFusion2 FPGA does have an RTC in the MSS but does not keep time with power-down situations.

This driver allows for full writing and updating of the time and date using the command-line parameters and returns the current time and date if request as text. The output text can be piped into another process or application program in a shell script if necessary.

*Typical return string:* **Local Time: 10:45:23 Local Date: 29-08-14 Current Day: friday\n**

### 14.13.4 LCD Controller

| | | |
|---|---|---|
| LCD Controller | *lcd_pcf5074.c* | LCD optional display controller of text and format |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/lcd_pcf5074.sh*

*Usage & Syntax*

lcd_pcf5074 parameter syntax (Must be 16 parameters with '-' switches):
-dev <addr> -sdabit <bit> -clkbit <bit> -gpioaddr <addr> -enbit <bit> -rsbit <bit> -init|-data <data>

| | |
|---|---|
| *-gpiooffs* | is the user memory offset within the UIO mechanism phys mem of the i2c LCD data |
| *-dev* | is the i2C address of the pcf5074 device on the i2c bus. Check circuit schematics |
| *-sdabit* | is the bit position within the i2caddr location of the sda signal. Zero based 0-31 |
| *-clkbit* | is the bit position within the i2caddr location of the clk signal. Zero based 0-31 |
| *-enbit* | is the bit position within the location of the LCDen sig. Zero based 0-31 |
| *-rsbit* | is the bit position within the location of the LCDrs sig. Zero based 0-31 |
| *-data* | is the string of data for the LCD to display. Expressed in HEX characters two per char |
| *-init* | is only included if LCD is to be initialized at first use (use only instead of -data) |

**Example** (init): lcd_pcf5074 -dev 60 -gpiooffs $GPIO_ADDR -sdabit 6 -clkbit 8 -enbit 14 -rsbit 11 –init

**Example** (data): lcd_pcf5074 -dev 60 -gpiooffs $GPIO_ADDR -sdabit 6 -clkbit 8 -enbit 14 -rsbit 11 –data (data here)

*Support: www.nineways.co.uk; pbates@nineways.co.uk*

*Description*

The LCD screen can be set to display any alpha-numeric character on a 16x2 mono-chrome orientation. The data and the settings are set as command-line parameters, and as an alternative to sending data, the screen can be initialized using the *–init* switch operator. As the screen contents survives a reboot (*but not a power down*) in terms of data content displayed, a manual initialization command has to be issued if you want to blank or revert back to default settings.

The data is passed with the *–data* switch and is entered in the form of printable HEX characters to allow for the full range of shapes and extended characters the screen supports without upsetting the uCLinux parameter restrictions.

### 14.13.5 General I/O (GPIO)

| | | |
|---|---|---|
| General I/O (GPIO) | *pcf5074.c* | Individual spare pins and the LEDs, Switches & Jumpers |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/pcf5074.sh*

*Usage & Syntax*

pcf5074 parameter syntax (Must be more than 12 parameters with '-' switches):
*-gpiooffs <addr> -sdabit <bit> -clkbit <bit> -gpio1 <string> -gpio2 <string> -gpio3 <string>*

| | |
|---|---|
| *-gpiooffs* | is the user memory offset within the UIO mechanism phys mem of the i2c GPIO signal |
| *-sdabit* | is the bit position within the i2caddr location of the sda signal. Zero based between 0-31 |
| *-clkbit* | is the bit position within the i2caddr location of the clk signal. Zero based between 0-31 |
| *-gpio1* | is 8 bits long binary word representing the output state of the P port |
| *-gpio2* | is 6 bits long binary word representing the output state of the P port |
| *-gpio3* | is 1 bit long binary word representing the output state of the P port (Green LED) |

**Example:** pcf5074 -gpiooffs 1879048192 -sdabit 2 -clkbit 1 -gpio1 10101110 -gpio2 101000 -gpio3

*Support: www.nineways.co.uk; pbates@nineways.co.uk*

*Description*

All of the NetFusion PCB individual I/O lines that are not buses or protocols are under the umbrella of this GPIO user space driver application. The command-line parameters setup the settings and also set the output states of those I/O configured as outputs.

The return string is the state of those GPIO lines configured as inputs. The outputs are the expansion GPIO connector, the user LED, relays and the RS232 extended signals. The inputs signals include the rotary switches, option jumper selectors, LCD buttons and the expansion connector inputs.

*Typical return string:* **2, 128\n3, 64\n4: 23\n**

### 14.13.6 Line-OUT Stereo Audio

| | | |
|---|---|---|
| Line-OUT Stereo Audio | *spi_audio_line_out.c* | Stream digital audio to analogue Line-OUT audio |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/spi_audio_line_out.sh*

*Usage & Syntax*

spi_audio_line_out parameter syntax (Must be 10 parameters with '-' switches):
-rightaddr <rightaddr> -leftaddr <leftaddr> -samprate <samprate> -datapathL <datapathL> -datapathR <datapathR>

| | |
|---|---|
| *-rightaddr* | is used to define the right base address of the spi_audio_line_out in memory |
| *-leftaddr* | is used to define the left base address of the spi_audio_line_out in memory |
| *-samprate* | is the channel left/right frequency rate |
| *-datapathL* | is the string that accesses channel 0 inode file to allow data transfer |
| *-datapathR* | is the string that accesses channel 1 inode file to allow data transfer |

**Example**: spi_audio_line_out -rightaddr 8 -leftaddr 4 -samprate 4000 -datapathL <datapathL>
-datapathR <datapathR>

When driver loads, it will check for an Enable, Disable or Quit Program code.
Please enter e = enable, d = disable or q = quit program. Carriage return terminated

*Support: www.nineways.co.uk; pbates@nineways.co.uk*

*Description*

The line-OUT audio is stereo and so has two independent channels. Once the driver has been executed on the command-line with the settings passed at the same time as parameters, the arbitrary input streams are opened ready for transmitting the data to a DAC then out the analogue line circuitry.

The two input streams are not the standard input stream but in fact two independently declared file streams that can be written to by other applications in order to send the audio data. These are setup on the command-line parameters before the session begins and for the duration of the session the two input file streams receive the data and send to the DAC and Line-OUT circuitry.

The standard input stream is also used in addition to the two file streams. The standard input allows single character commands (*shown above*) to enabled, disabled the audio and also to exit the session and close the driver application. The file streams allow the user to create their own functionality to handle the sent audio data – maybe the user will write an application to receive from the UDP/IP using RTP and then send the audio digitized data on each of the independent input file streams.

**Important Note***: the data is 8-bit unsigned. This means that 128 is the mid-line silence and 0 and 255 are the negative and positive extremes of the audio signal. This is unlike signed values whereby zero is the mid-point. This driver operates as unsigned however.*

### 14.13.7 Line-IN Stereo Audio

| | | |
|---|---|---|
| Line-IN Stereo Audio | *spi_audio_line_in.c* | Line-IN audio analogue to digital output stream |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/spi_audio_line_in.sh*

*Usage & Syntax*

spi_audio_line_in parameter syntax (Must be 8 parameters with '-' switches):
-baseaddr <baseaddr> -samprate <samprate> -datapathL <datapathL> -datapathR <datapathR>

| | |
|---|---|
| *-baseaddr* | is used to define the base address of the spi_audio_line_in in memory |
| *-samprate* | is the channel left/right frequency rate |
| *-datapathL* | is the string that accesses channel 0 inode file to allow data transfer |
| *-datapathR* | is the string that accesses channel 1 inode file to allow data transfer |

**Example**: spi_audio_line_in -baseaddr 1073811456 -samprate 4000 -datapathL /dev/datapathL -datapathR /dev/datapathR

When driver loads, it will check for an Enable, Disable or Quit Program code
Please enter e = enable, d = disable or q = quit program. Carriage return terminated.

*Support: www.nineways.co.uk; pbates@nineways.co.uk*

*Description*

The line-IN audio is stereo and so has two independent channels. Once the driver has been executed on the command-line with the settings passed at the same time as parameters, the arbitrary output streams are opened ready for receiving the data from the ADC from the analogue line circuitry.

The two output streams are not the standard output stream but in fact two independently declared file streams that can be read from by other applications in order to receive the audio data. These are setup on the command-line parameters before the session begins and for the duration of the session the two output file streams send the data to the piped application from the ADC and Line-IN circuitry.

The standard input stream is also used in addition to the two file streams. The standard input allows single character commands (*shown above*) to enabled, disabled the audio and also to exit the session and close the driver application. The file streams allow the user to create their own functionality to handle the received audio data – maybe the user will write an application to send to the UDP/IP using RTP which was received from the audio digitized data on each of the independent output file streams.

**Important Note***: the data is 8-bit unsigned. This means that 128 is the mid-line silence and 0 and 255 are the negative and positive extremes of the audio signal. This is unlike signed values whereby zero is the mid-point. This driver operates as unsigned however.*

### 14.13.8 PCB Temperature Sensor

| | | |
|---|---|---|
| PCB Temperature Sensor | *temp_sensor.c* | An on-board temperature sensor on the baseboard |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/temp_sensor.sh*

*Usage & Syntax*

temp_sensor parameter syntax :

-file <filename> is optional and if present on the command line parameter list, saves the output from the driver to a file rather than to the standard output stream

**Example**: temp_sensor –file /usr/temp

*Support: www.nineways.co.uk; pbates@nineways.co.uk*

*Description*

The temperature sensor on-board the NetFusion PCB baseboard outputs the real-time temperature. When launching this driver, it operates until a command in the standard input stream forces the driver application to exit. Whilst running, the calculated, averaged and settled temperature is output once every 60 seconds. With the absence of a declared output file, the temperature string is written to the standard output stream. However, if a file is passed as a command-line parameter, then the temperature string is outputted to the file every 60 seconds instead.

*Typical return string:* **+55.4C\n**
*Typical return string:* **-2.3C\n**

### 14.13.9 Voltage PCB Supervisor

| | | |
|---|---|---|
| Voltage PCB Supervisor | *vs_ad7998.c* | Monitor and check all voltages and rails on the PCB |

**This driver is demonstrated and handled in the shell script on NetFusion**: */drivers/vs_ad7998.sh*

*Usage & Syntax*

vs_ad7998 parameter syntax (Must be 10 parameters with '-' switches):
-dev <addr> -sdabit <bit> -clkbit <bit> -gpioaddr <addr> -convbit <bit>

| | |
|---|---|
| *-gpiooffs* | is the user memory offset within the UIO mechanism phys mem of the i2c GPIO signal |
| *-dev* | is the i2C address of the ad7998 IC device on the i2c bus. Check circuit schematics |
| *-sdabit* | is the bit position within the i2caddr location of the sda signal. Zero based between 0-31 |
| *-clkbit* | is the bit position within the i2caddr location of the clk signal. Zero based between 0-31 |
| *-convbit* | is the bit position within the convaddr location of the conv signal. Zero based between 0-31 |

**Example**: vs_ad7998 -dev 33 -gpiooffs 1879048192 -sdabit 5 -clkbit 4 -convbit 18

*Support: www.nineways.co.uk; pbates@nineways.co.uk*

*Description*

The NetFusion baseboard has many different power supply voltages and rails. This driver makes it possible for the application code to gain access to the power voltage levels via an IP core and an ADC device in the hardware.

By calling and invoking this application driver, the command-line parameters provide the settings and then the return text provides the voltage levels. Like the GPIO, RTC and the ACI driver *(above),* this driver exits immediately with the values and does not maintain continuous operation. To read the voltages again, this driver must be invoked once again and the new updated values are output as text.

*The output text displays the intended voltage in the order:* **5V, 2.5V, 1.8V, 3.3V (SOM), 3.3V (Baseboard), 1.2V, Vin (Power DC inlet)**

*Typical return string:* **4.97V, 2.39V, 1.81V, 3.28V, 3.26V, 1.19V, 18.37V\n**

# 15 Running a SoftConsole "Hello, World!" Project on NetFusion

This application note explains how to use the Libero SoC and SoftConsole software to develop a simple Application on the M2S-FG484 NetFusion baseboard. The application is a "Hello, world" demo that uses GPIO to blink the on-board LED and use the LPDDR memory for data storage.

## 15.1 Hardware Platform

The hardware platform is the NetFusion baseboard. The demo described in this application note assumes that the mini-USB cable is plugged into the mini-USB connector on the baseboard (to provide the serial console) and the FlashPro4 (or FlashPro3) JTAG programmer is connected also. The serial connection may require the *FT232R* USB drivers to be installed to the development host PC. Also, please open the serial console window on the USB-Serial console port using any terminal application (*HyperTerm, PuTTY*, etc.) with the 115200,8N1 parameters.

## 15.2 Installing the Software on the Development Host PC

For this application note we use SoftConsole and Libero software running on a Windows development host PC.

Download and install Libero SoC v11.4. Select the following components during installation:

- Synplify Pro
- SoftConsole 3.4
- ModelSim AE with SmartFusion and SmartFusion2

Download the NetFusion M2S-FG484 Starter Libero project and open it in Libero, update IP cores if asked.

Download the prebuilt NetFusion M2S-FG484 firmware design [.stp] file. Program the [.stp] design into M2S-FG484 SOM using the FlashPro:

- Start FlashPro on a Windows host PC.
- From the FlashPro IDE, create a new project with an arbitrary name.
- From the main FlashPro window, push **Configure Device**.
- Push **Browse next** to load existing programming file.
- Browse to the **uCLinux** SmartFusion2 project *[.stp]* file and choose it.
- Power reset the board after the successful programming.

### 15.3 Running and Debugging the Project

In the **Libero Design Flow** view, launch the **Write Application Code** action. Soft Console will start with the sample project workspace with two projects opened.

Start debugging the prebuilt application. Open the **Debug as** pull-down menu (to the right of the **bug**button) and select g4m_system_MSS_CM3_app Debug. Confirm the **Perspective Launch** dialog.

The program will be loaded to the NetFusion SRAM and stopped on the first breakpoint at *main().*

Step over (F6) and into the functions (F5), examine variables (just point the mouse over the variable), and so on. Single-step until up to the *sprintf()* just after the *config_mddr_lpddr().*

Add a memory monitor to the RAM address 0xa0000000 using the **Memory** tab (lower SoftConsole window). Now step over *sprintf()* and observe that the string appears at location 0xa0000000.

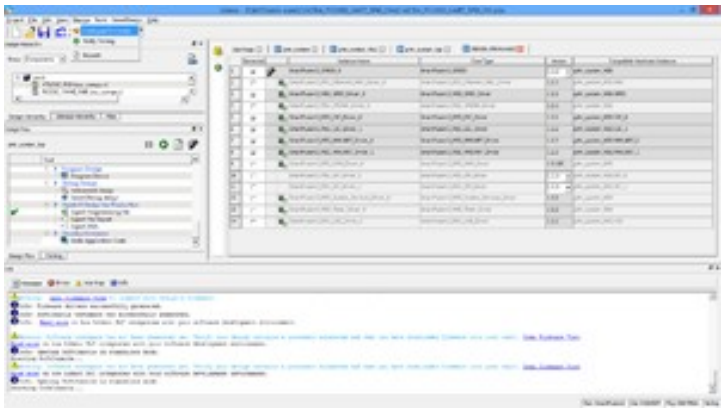
Resume the program run with F8 ("green triangle" in the **Debug** window). Observe the SoftConsole application working on the NetFusion baseboard.

### 15.4 Modifying the Libero and SoftConsole Projects

To modify the Libero project, rebuild and rerun the application in SoftConsole:

Terminate the program with Ctrl-F2 ("red square") and switch back to the **C/C++ Editor** perspective. Don't leave SoftConsole.

In Libero, open the **Design -> Configure Firmware** tab. Let's add a new **software driver** to our SoftConsoleproject. Add a checkbox to the SmartFusion2_MSS_NVM_Driver_0. Re-generate SoftConsole workspace (yellow barrel with gear in the middle of the Libero window).



Now switch to SoftConsole and observe that the *mss_nvm* firmware appears in the tree of the g4m_system_MSS_CM3_HW_platform project (in the *drivers/*directory).

Before rebuilding the project, open the sys_config.h *(present in the editor tab)* and change.

```
#define MSS_SYS_MDDR_CONFIG_BY_CORTEX        1
```

To

```
#define MSS_SYS_MDDR_CONFIG_BY_CORTEX        0
```

This is needed because Libero re-generated the SoftConsole project workspace and unconditionally enabled initialization of the LPDDR memory by the DDR module firmware added to the SoftConsole project. This initialization is incompatible with the memory used on the NetFusion M2S-FG484 SOM (the demo application uses the correct LPDDR initialization code in *config_mddr_lpddr()).*

- Rebuild project: **Project -> Clean**, select **Clean all projects** and **Start the build immediately**.
- Re-run the debug session of the project.

# 16 Installing a Bootable uCLinux Image to the NetFusion Board

Step through the following procedure in order to install the *netfusion.uImage* bootable uCLinux image to your board:

- Download the bootable uCLinux image (*netfusion.uImage*) and copy it to the *TFTP* boot directory on your development host PC *(C:\Program Files\Tftpd32 if you installed Tftpd32 on the Windows host PC)*.

- Connect to the serial console, reset the board and stop auto-boot by hitting any key.

- Being at the U-boot prompt, run the following U-boot commands to load the image over *TFTP* and install it to the flash:

```
BOARD> setenv image netfusion.uImage
BOARD> run update
```

Next reset will load the newly installed image from the flash:

```
BOARD> reset
resetting ...
```

# 17 Booting uClinux on SmartFusion2 in 0.5 Seconds

## 17.1 Fast-booting uCLinux on SmartFusion2

Watch this video to see how Linux (*uCLinux*) boots up on SmartFusion2 from power-up to the interactive shell in about 0.5 seconds.

Both Cortex-M3 and LPDDR are configured to run at 166MHz. PCLK0 is set to a 1/2 of the main clock (83MHz), allowing to run SPI_0 and access the SPI flash at up to 41.5MHz. The on-chip cache is enabled by software for LPDDR.

The uCLinux configuration used in this demo targets a typical "data harvesting" device. Support for the SPI controller, SPI flash, flash partitioning and JFFS2 are all enabled in the kernel. As soon as uCLinux comes up to the shell, a data harvesting application (or any number of such applications) can be started from the */etc/rcstart-up* script, collecting data from various sensors and logging collected data into the system-on-module SPI flash. As can be seen from the above **video**, it takes uClinux only 0.5 seconds from power-up / reset to the point where data harvesting can commence.

## 17.2 uCLinux Bootstrap Sequence

The uCLinux bootstrap can be described by the following sequence:

- On reset, U-boot firmware runs from the on-chip eNVM / eSRAM. U-boot performs SmartFusion2 initialization from power-on / reset, including setting up the DDR controller and enabling access to the SPI flash.
- U-boot copies the uCLinux bootable image from the SPI flash to the LPDDR and then passes control to the kernel entry point.
- uCLinux proceeds to boot up in LPPDR. A RAM-based filesystem *initramfs* is mounted as a root filesystem. *initramfs* is populated with required files and directories and is then simply linked into the uCLinux bootable image at build time on a cross-development host PC.
- As a part of the boot-up sequence, the SPI controller, SPI flash device drivers, and the JFFS2 sub-system are initialized by the kernel.
- Having mounted the root file system, uCLinux starts the init application, which in turn spawns the first shell and runs a start-up command sequence defined in the */etc/rc script*.
- */etc/rc* can do, generally speaking, whatever. In this demo, it is assumed that */etc/rc* will be defined to start a data harvesting application as soon as possible. To emulate this, we run an echo “Data harvesting can commence" command from the start-up script.

## 17.3 Installing the Demo

The procedure described here explains how to install the bootable uCLinux image (*fastboot.uImage*) to the NetFusion.

Here is how you can build and install the bootable uCLinux image from the project sources (*fastboot.tgz*), having installed them on top of the NetFusion SmartFusion2 uClinux distribution.

***Note:*** *The uCLinux image and the sample project have been built and validated in context of the uCLinux Release 1.11.0. If you are using a different release, some porting changes may be needed.*

## 17.4 Understanding the Demo

The first message printed by U-boot after power-up/reset is as follows:

```
U-boot 2010.03-linux-cortexm-1.10.0 (May 17 2013 - 16:09:15)
```

U-boot proceeds with initialization of the NetFusion and reports various characteristics of the *hardware platform:*

```
CPU : SmartFusion2 cSoC (Cortex-M3 Hard IP)
Freqs: CORTEX-M3=166MHz,PCLK0=83MHz,PCLK1=83MHz
Board: M2S-FG484-SOM Rev 1A, www.emcraft.com
DRAM: 64 MB
In: serial
Out: serial
Err: serial
Net: M2S_MAC
```

U-boot then waits for the number of seconds defined by the *bootdelay* environment variable before proceeding to execute a bootup sequence defined by the *bootcmd* variable. If the user presses any key while in the *bootdelay* loop, U-boot interrupts the boot-up sequence and enters the interactive command monitor. Initially we set *bootdelay* to 3.

```
Hit any key to stop autoboot: 0
```

Now, this is the point when the *bootdelay* loop has expired and the *bootcmd* commands sequence is being executed. On SmartFusion2, the first *action bootcmd* performs is loading a bootable uCLinux image from the SPI flash to the LPDDR:

```
16384 KiB S25FL128S_64K at 0:0 is now current device
```

Then a *bootm* command is called upon the image loaded into DDR. What this command does is, parse the image header, figure out the image type and size, optionally verify the checksum and pass control to the kernel entry point:

```
## Booting kernel from Legacy Image at a0007fc0 ...
Image Name: Linux-2.6.33-arm1
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 941920 Bytes = 919.8 kB
Load Address: a0008000
Entry Point: a0008001
```

This is where U-boot is done and the uCLinux kernel starts running:

```
Starting kernel ...
```

The following are the first messages printed by the kernel:

```
Linux version 2.6.33-arm1 (vlad @ocean.emcraft.com) (gcc version 4.4.1 (Sourcery G++ Lite 2010q1-
189) ) #79 Thu Dec 27 11:32:17 MSK 2012
CPU: ARMv7-M Processor [412fc231] revision 1 (ARMv7M)
CPU: NO data cache, 8K instruction cache
Machine: Actel M2S
...
```

Note the kernel input arguments passed by U-boot to the kernel. This is defined by *bootargs* in U-boot:

```
Kernel command line: m2s_platform=m2s-fg484-som console=ttyS0,115200 panic=10
ip=172.17.4.219:172.17.0.1:::m2s-fg484-som:eth0:off ethaddr=C0:B1:3C:83:83:83
...
```

The kernel proceeds to set the memory up. We have 64 MB of LPDDR to run uCLinux from:

```
Memory: 64MB = 64MB total
Memory: 64016k/64016k available, 1520k reserved, 0K highmem
Virtual kernel memory layout:
vector : 0x00000000 - 0x00001000 ( 4 kB)
fixmap : 0xfff00000 - 0xfffe0000 ( 896 kB)
vmalloc : 0x00000000 - 0xffffffff (4095 MB)
lowmem : 0xa0000000 - 0xa4000000 ( 64 MB)
modules : 0xa0000000 - 0x01000000 (1552 MB)
.init : 0xa0008000 - 0xa0047000 ( 252 kB)
.text : 0xa00cd7e8 - 0xa00e3000 ( 87 kB)
.data : 0xa00e4000 - 0xa00edf60 ( 40 kB)
...
```

Then the kernel runs the so-called "calibration loop" in order to calculate loops-in-jiffies also referred to as lpj. The lpj value is used by the kernel to implement various delays. Note that as part of the calibration loop, the kernel also calculates the *BogoMIPS* value, which is a uCLinux approximation of the MIPS (Million Instructions Per Second) metric:

```
Calibrating delay loop... 154.82 BogoMIPS (lpj=774144)
...
```

The UART device driver and the serial console are initialized:

```
Serial: 8250/16550 driver, 2 ports, IRQ sharing disabled
serial8250.0: ttyS0 at MMIO 0x40000000 (irq = 10) is a 16550A
console [ttyS0] enabled
```

The SPI device driver is started and the SPI flash is partitioned onto three separate logical partitions. Each partition can be accesses as a separate logical device by user-space code:

```
m25p80 spi0.0: s25fl129p1 (16384 Kbytes)
Creating 3 MTD partitions on "s25fl129p1":
0x000000000000-0x000000010000 : "spi_flash_uboot_env"
0x000000010000-0x000000410000 : "spi_flash_linux_image"
0x000000410000-0x000001000000 : "spi_flash_jffs2"
spi_m2s spi_m2s.0: SPI Controller 0 at 40001000,clk=83000000
...
```

The *initramfs* root filesystem is mounted and init, which is the first user-space process, is launched:

```
init started: BusyBox v1.17.0 (2012-12-14 10:29:39 MSK)
```

*init* proceeds to execute a start-up command-line sequence defined by the */etc/rc script*. In our application, we have */etc/rc* contain a single command (echo "Data harvesting can commence"), which results in the output below:

```
Data harvesting can commence
```

And finally, init starts the interactive command shell:

```
/ #
```

### 17.5 Configuring for Fast-Boot

To achieve the ~0.5 seconds boot times shown in the above video, you will need to specifically configure U-boot. Note that each of the configuration steps described below is optional, in the sense that if you don't perform a certain optimization step, the demo will still boot on the NetFusion, although slightly slower than it would otherwise.

*From the U-boot prompt, do the following:*

Disable validation of the bootable uCLinux image checksum:

```
M2S-FG484-SOM> set verify no
```

Add lpj=774144 to *bootargs*. This tells the kernel to avoid running the calibration loop and instead set lpj (loops-per-jiffy) to a fixed value (pre-calculated for the specific hardware configuration):

```
M2S-FG484-SOM> set bootargs m2s_platform=m2s-fg484-som console=ttyS0,115200 panic=10
lpj=774144
```

Add quiet=quiet to *bootargs* in U-boot. This tells the kernel to suppress printing information messages onto the serial console.

```
M2S-FG484-SOM> set bootargs m2s_platform=m2s-fg484-som console=ttyS0,115200 panic=10
lpj=774144 quiet=quiet
```

To restore the message printing function, **remove** quiet=quiet from *bootargs*.

Set *bootdelay* to 0. This tells U-boot to immediately proceed with running *bootcmd,* without waiting for user input for *bootdelay* seconds:

```
M2S-FG484-SOM> set bootdelay 0
```

To restore the *bootdelay* function in U-boot, press Ctrl-C at the same time as you push the reset button. This will return you to the U-boot command monitor, where you can reset *bootdelay* to a non-zero value.

Save the updated environment variables to the SPI flash:

```
M2S-FG484-SOM> saveenv
Saving Environment to SPI Flash...
Erasing SPI flash...Writing to SPI flash...done
```

Push the reset button and watch the demo to boot up on SmartFusion2 in a blink of eye:

```
U-Boot 2010.03-00001-gc9f691b (Dec 26 2012 - 14:06:31)

CPU : SmartFusion2 SoC (Cortex-M3 Hard IP)
Freqs: CORTEX-M3=166MHz,PCLK0=83MHz,PCLK1=83MHz
Board: M2S-FG484-SOM Rev 1A, www.emcraft.com
DRAM: 64 MB
In: serial
Out: serial
Err: serial
Net: M2S_MAC
Hit any key to stop autoboot: 0
16384 KiB S25FL128S_64K at 0:0 is now current device
## Booting kernel from Legacy Image at a0007fc0 ...
Image Name: Linux-2.6.33-arm1
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 941920 Bytes = 919.8 kB
Load Address: a0008000
Entry Point: a0008001
Loading Kernel Image ... OK
OK

Starting kernel ...

init started: BusyBox v1.17.0 (2012-12-14 10:29:39 MSK)
Data harvesting can commence
/ #
```

## 17.6 Restoring Default Configuration

Please step through the following procedure to restore the default configuration:

Push and hold ^C. At the same time, push the reset button on the NetFusion baseboard. This will bring you to the U-boot prompt:

```
U-Boot 2010.03-00001-gc9f691b (Dec 26 2012 - 14:06:31)
CPU : SmartFusion2 SoC (Cortex-M3 Hard IP)
Freqs: CORTEX-M3=166MHz,PCLK0=83MHz,PCLK1=83MHz
Board: M2S-FG484-SOM, www.emcraft.com
DRAM: 64 MB
In: serial
Out: serial
Err: serial
Net: M2S_MAC
Hit any key to stop autoboot: 0
M2S-FG484-SOM>
M2S-FG484-SOM>
M2S-FG484-SOM>
M2S-FG484-SOM>
M2S-FG484-SOM>
```

Roll back the changes you have recently made:

```
M2S-FG484-SOM> print bootargs
bootargs=m2s_platform=m2s-fg484-som console=ttyS0,115200 panic=10 lpj=774144 quiet=quiet
M2S-FG484-SOM> setenv bootargs m2s_platform=m2s-fg484-som\
console=ttyS0,115200 panic=10
M2S-FG484-SOM> setenv bootdelay 3
M2S-FG484-SOM> setenv verify
M2S-FG484-SOM> saveenv
Saving Environment to SPI Flash...
Erasing SPI flash...Writing to SPI flash...done
M2S-FG484-SOM>
```

## 18 Using DHCP Client in U-boot

Step through the following procedure in order to add *DHCP* client to U-boot:

Define the **CONFIG_CMD_DHCP** configuration parameter in the U-boot configuration file for your board (*u-boot/include/configs/<board>.h*):

```
...
#define CONFIG_CMD_DHCP
...
```

Refer to the *u-boot/README.u-boo*t file and the *u-boot/doc/* directory for more *DHCP*-related options.

Build the U-boot image as described in this document (*previously earlier).*

Upgrade U-boot on your board.

Connect to the serial console, reset the NetFusion baseboard and stop auto-boot by hitting any key. Being at the U-boot prompt, run the *dhcp* command to invoke *DHCP* client to obtain an IP address and other network parameters:

```
BOARD> dhcp
```

# 19 Setting the Stack Size for a User-Space Application

This application note explains how to set up the stack size for a user-space application.
The stack size can be changed using the *arm-uclinuxeabi-flthdr* utility:

```
[psl@ocean app]$ arm-uclinuxeabi-flthdr -?
arm-uclinuxeabi-flthdr: invalid option -- '?'
invalid option
usage: arm-uclinuxeabi-flthdr [options] flat-file
Allows you to change an existing flat file

-p : print current settings
-z : compressed flat file
-d : compressed data-only flat file
-Z : un-compressed flat file
-r : ram load
-R : do not RAM load
-k : kernel traced load (for debug)
-K : normal non-kernel traced load
-s size : stack size
-o file : output-file
(default is to modify input file)
Report bugs to <URL:https://support.codesourcery.com/GNUToolchain/>
[psl@ocean app]$ arm-uclinuxeabi-flthdr app
app
Magic: bFLT
Rev: 4
Build Date: Fri Nov 23 13:24:57 2012
Entry: 0x45
Data Start: 0x4300
Data End: 0x5440
BSS End: 0x7500
Stack Size: 0x1000
Reloc Start: 0x5440
Reloc Count: 0x7d
Flags: 0x1 ( Load-to-Ram )
[psl@ocean app]$ arm-uclinuxeabi-flthdr -s 8192 app
[psl@ocean app]$ arm-uclinuxeabi-flthdr app
app
Magic: bFLT
Rev: 4
Build Date: Fri Nov 23 13:24:57 2012
Entry: 0x45
Data Start: 0x4300
Data End: 0x5440
```

```
BSS End: 0x7500
Stack Size: 0x2000
Reloc Start: 0x5440
Reloc Count: 0x7d
Flags: 0x1 ( Load-to-Ram )
[psl@ocean app]$
```

# 20 Building Multi-Threaded Applications

The version of the uClibc included with the CodeSourcery tool chain does not provide support for POSIX threads. To address this limitation, we have enabled POSIX threads in uClibc and included the updated library built for Thumb in our uClinux distribution.

*In order to build a multi-threaded application and link it with the appropriate uClibc binary, do the following:*

After activation of the cross-build environment (*source ACTIVATE.sh*), run the following command (*linux-dp.c* being the application in the below example):

```
arm-uclinuxeabi-gcc -o linux-dp linux-dp.c -I \
${INSTALL_ROOT}/A2F/root/usr/include -mcpu=cortex-m3 -mthumb -L \
${INSTALL_ROOT}/A2F/root/usr/lib -pthread
```

# 21 How to Define an IRQ Handler in a Custom uCLinux Device Driver

This note explains how to add an IRQ handler to a custom device driver in the uClinux kernel running on the SmartFusion2 SOM on NetFusion.

The specific example below was developed on the M2S-FG484 System-On-Module plugged into the NetFusion baseboard. The example defines an IRQ handler for the user push button. The push button input is tied to MSS GPIO_0 in the default Libero design used with the M2S-FG484 SOM. This is IRQ 50 in the NVIC space.

Here is the code that registers the IRQ and enables the interrupt at the GPIO level:

```
/*
* Register an IRG handler for the User-push button press event
*/
ret = request_irq(50, sample_irq, 0, "sample", 0);
if (ret < 0) {
printk(KERN_ALERT "%s: request_irg failed with %d\n",
__func__, ret);
goto Done;
}

/*
* Configure GPIO_O as input, enable interrupts, Level Low
*/
* (unsigned int *) (0x40013000 + 0) = (1<<1) | (1<<3) | (1<<5);
```

Here is the interrupt handler. For the level sensitive interrupts, no explicit clearing is needed, for edge sensitive you would have to add code to explicitly clear the interrupt by writing into the GPIO_IRQ register:

```
static irqreturn_t sample_irq(int irq, void *dev_id)
{
printk("irq %d\n", irq);

return IRQ_RETVAL(1);
}
```

Here is how the code was tested on the NetFusion.
Load the driver as a kernel module (the driver was developed over NFS):

```
/mnt # insmod sample.ko
/mnt #
```

Push the button, and then release it quickly. As can be seen from the snapshot below, the interrupt is triggered as long as there is the hardware event (the push button signal is being low), then there is no interrupt when the push button is released:

```
/mnt # irq 50
irq 50
irq 50
...
irq 50
/mnt #
/mnt #
/mnt # ls
Makefile            app.c           sample.c        sample.mod.o
Module.symvers      app.gdb         sample.ko       sample.o
app                 modules.order   sample.mod.c
/mnt #
```

# 22 Understanding what causes SEGV in an Application

If a user-space application terminates with SEGV, this means, as it is always the case in uCLinux, that the application did something that triggered an exception to the Cortex-M processor core. The most likely causes are:

- You built your application for a processor architecture other than Cortex-M, which causes an "Unsupported instruction" exception.
- Your application makes an attempt to access an address outside of the Cortex-M address space.
- Your application makes an attempt to access unaligned data.

To understand what is causing an exception, do the following:

Enable **CONFIG_DEBUG_USER** in the kernel configuration. From your project directory, bring up the kernel configuration interface by running make *kmenuconfig*. Go to Kernel hacking and *enableVerbose* user fault messages.

In U-boot, add *user_debug=8* to *bootargs*.

Having done the above, rebuild your uCLinux project, bring it up on the NetFusion and re-run your application. When the application fails with *SEGV*, there should be some informational messages on the system console indicating the cause of the fault.

# 23 References

Visit Emcraft's web site at www.emcraft.com to obtain additional materials related to Linux Cortex-M.

Emcraft Systems provides detailed kit information and materials from this page:
http://www.emcraft.com/som/m2s

As new SmartFusion2 and kit related materials become available, they will be posted on the Microsemi and Emcraft Systems web sites. Microsemi recommends that you sign up for Product Updates to be notified when new material is available.

Microsemi product updates: https://www.actel.com/portal/default.aspx?r=2

Emcraft Systems product updates: http://www.emcraft.com/som/m2s/register

Refer to *Emcraft Systems SmartFusion2 SOM (System-On-Module) Hardware Architecture* for detailed information on the hardware architecture of the Emcraft Systems SmartFusion2 SOM board.

Refer to *Emcraft Systems SOM-BSB-EXT Baseboard Hardware Architecture* for detailed information on the hardware architecture of the Emcraft Systems SOM-BSB-EXT baseboard.

Refer to *Linux Cortex-M User's Manual* for detailed information on the software architecture of the Linux SmartFusion2 distribution.

## 24 Contact


| | |
|---|---|
| **Nine Ways Research & Development Ltd** | E-Mail : **pbates@nineways.co.uk**<br>Internet : **www.nineways.co.uk**<br>**UK**<br>Unit G.15, iDCentre, Lathkill House, rtc Business Park<br>London Road, Derby. DE24 8UP<br>United Kingdom<br>Tel : **+44 (0) 1332 258847**<br>FAX : **+44 (0) 1332 258823** |

# 25 Document History

Document Change Notices (DCO)

| Version | Description | Created/Changed By | Date |
|---|---|---|---|
| Version 1.0 | Initial Release according to Version 1.0 | Paul Bates: Nine Ways | **29th August 2014** |

*Copyright © Nine Ways R&D Ltd 2014. All Rights Reserved.*

**Table 6: Document History Entry Log**